Coordinated Negotiated Search: A Generic Framework for Distributed Planning, Scheduling and Resource Allocation

Final Technical Report
Period: 5/15/92-5/14/95

Victor R. Lesser, ***Principal Investigator***

Distributed Artificial Intelligence Laboratory
Computer Science Department, Box 34610
Lederle Graduate Research Center
University of Massachusetts
Amherst, MA 01003-4610
(413) 454-1322

Sponsored by Advanced Research Projects Agency
ARPA Order No. 8744
Grant Number: N00014-92-J-1698
Contract Period: 5/15/92–5/14/95
Contract Dollars: $927,729

CONTENTS

1 NUMERICAL PRODUCTIVITY MEASURES

Refereed papers submitted but not yet published: 2

Refereed papers published: 42

Unrefereed reports and articles: 17

Books or parts thereof submitted but not yet published: 2

Books or parts thereof published: 2

Patents filed but not yet granted: 0

Patents granted: 0

Invited presentations: 8

Contributed presentations: 58

Honors received: 30

Prizes or awards received: 0

Promotions obtained: 3

Graduate Students supported: 8

Post-docs supported: 4

Minorities supported: 0

2 EXECUTIVE SUMMARY

Computer-based, decentralized decision making involving planning, scheduling, and resource allocation (DPSRA) problems is increasingly important as we attempt to create agile military and commercial organizations that can exploit the enormous amount of information that is available on-line and the emerging capability for on-line organizational interaction (e.g., enterprise integration systems, the electronic marketplace, etc.). Examples of DPSRA problems include logistical resource scheduling, crisis management, and concurrent engineering. The design of such applications is fraught with difficulties because agents in such systems cannot independently avoid conflicts, cannot access a global perspective to schedule their actions, cannot easily search for solutions in isolation, cannot respond statically to real-time deadlines, and must cope with an uncertain and changing environment. A major hurdle facing the construction of DPSRA applications is the lack of a generic framework for solving the difficulties outlined above. This generic framework will make it possible to significantly speed up the development of future DPSRA applications.

Central to our approach is the creation of a Coordinated Negotiated Search (CNS) framework—one that views negotiation and coordination as integral parts of the cooperative search process for a solution mutually acceptable to all agents. This framework integrates a wide range of negotiation strategies for different situations. These strategies are based on a sophisticated view of negotiation as a multi-level, multi-stage, and multi-anchored process in which agents not only exchange domain proposals and critiques but also exchange meta-level information about their dynamically evolving local and composite search spaces. These strategies do not depend solely on centralized mediation, or unlimited communication or computational resources, or on agents having homogeneous structures or representations. DPSRA systems will not work on only one problem at a time, but rather on a continually evolving set of interrelated problems. Coordination strategies are based on domain independent coordination relationships among tasks. This approach clearly delineates the coordination component of a distributed agent from the agent's local scheduling mechanisms. Strategies for coordination (of problem solving, negotiation, and monitoring) have been modeled and analyzed based on the quantitative properties of these coordination relationships and of other characteristics of the environment.

The specific applications we have used to exemplify the coordination and negotiation issues of DPSRA systems are airline terminal resource scheduling (gates, fuel trucks, bag trucks, etc.), multi-depot vehicle routing, and cooperative agent design of steam condensers. Recent work, though not yet completely implemented, has involved a cooperative information gathering application running on the Internet. Additionally, a sophisticated simulation system, called TAEMS, has also been constructed for testing the effectiveness of different coordination strategies.

3 SUMMARY OF TECHNICAL RESULTS

The following represent the major technical accomplishments of the contract:

• Development of GPGP, the first domain-independent architecture for distributed, real-time agent coordination.

A family of generic, real-time distributed coordination algorithms for use with cooperative agents has been developed, called GPGP. This family allows for a wide range of cooperation strategies, tailored to the needs of the specific application environment, to be implemented within a simple and extensible framework. This framework makes a clear separation among the coordination module, the local real-time scheduler and the application program. This is important because in many real applications we do not want to replace the existing application, but rather improve its performance by applying coordination techniques. Thus, these coordination mechanisms must interact smoothly with existing system components.

- Development of TAEMS, a simulation system and formal language for studying agent coordination issues.

As part of the effort in developing GPGP, we have also completed a formal framework, TAEMS (Task Analysis, Environment Modeling, and Simulation), for specifying task environments and analyzing coordination algorithms with respect to multiple performance criteria, and we have implemented a simulator. TAEMS allows users to do both exploratory research into possible performance effects and to verify analytically derived models of the effects of environmental characteristics on coordination algorithm performance. We used this framework to build a model of a simplified, distributed interpretation task, and examined the question of how various organizations of agents would perform in this environment, building an analytic model for describing the performance of three coordination algorithms. It has also been applied recently to a problem involving NASA regarding how to coordinate its proposed distributed data analysis centers.

- Development of TEAM, a reusable agent architecture for concurrent engineering design and its realization in a sophisticated design application involving seven expert systems.

To support the integration of heterogeneous and reusable agents into functional agent sets, a multiagent framework, TEAM, has been implemented. Conflict is an integral part of problem solving in multi-agent systems and is often the focal point of interaction among agents. Our work acknowledges conflict as a driving force in the control of distributed-search activity. The effectiveness of this architecture has been investigated in a seven-agent steam condenser design system. This system outperforms an existing mechanical design system that exploits the same knowledge but is not structured as a multiagent negotiation process.

- Development of DARM, a complex distributed scheduling system involving the scheduling of resources at an airport.

The DARM (Distributed Airport Resource Manager) distributed scheduling application has been used to verify and extend earlier work by Sycara et al. on the importance and role of meta-level information in achieving efficient distributed scheduling. A testbed has been created that can be configured as a community of two or more agents, each with its own resources (i.e., gates, fuel trucks, baggage handlers) and each responsible for satisfying its own schedule of arriving and departing flights. The need for cooperation and negotiation arises because an individual agent may lack sufficient resources to satisfy its schedule and may have to borrow these resources from other agents. By coordinating the individual scheduling efforts so that each agent understands the probable requirements of other agents, the likelihood increases that remote agents will be able to lend the appropriate resource at the time it is required. In the event that no globally satisfactory solution can be found, agents must negotiate in order to determine which local constraints can be relaxed to enable such a solution to be developed. The presence of meta-level information allows agents to more accurately determine whether to solve subproblems locally (through backtracking and constraint relaxation) or whether to apply to other agents for resources. This system is the most sophisticated distributed scheduling application developed to date and therefore represents an important data point in assessing the feasibility of a distributed scheduling approach for real applications. A side benefit of this effort has been the development of the

DSS domain independent job shop scheduling system whose performance benchmarks are comparable or better than existing systems.

- Development of a new negotiation protocol for electronic commerce; the advantages of this protocol over the contract-net protocol have been formally justified.

We have significantly extended the original contract-net negotiation protocol for use with self-interested agents involved in electronic commerce. An important aspect of this protocol is that it allows for a contract to specify a unilateral decommitant penalty. We have shown that this capability improves expected social welfare and Pareto efficiency of contracts by allowing better accommodation of future events.

Based on these technical accomplishments and the verification of the usefulness of these approaches both empirically through implementing them on complex applications and through formal analytic techniques, we have made significant progress in our goal of developing generic architectures appropriate for distributed planning, scheduling, and resource allocation (DPSRA) problems.

4 PUBLICATIONS, REPORTS AND ARTICLES

4.1 Refereed Papers Accepted but not yet Published

Lander, S. and Lesser, V. "Sharing Meta-Information to Guide Cooperative Search Among Heterogeneous Reusable Agents," to appear in *IEEE Transactions on Knowledge and Data Engineering.*

NagendraPrasad, M.V., Lesser, V. and Lander, S. "Reasoning and Retrieval in Distributed Case Bases," to appear in *Journal of Visual Communication and Image Representation, Special Issue on Digital Libraries.*

4.2 Refereed Papers Published

Decker, K. and Lesser, V. "Coordination Assistance for Mixed Human and Computational Agents." In *Proceedings of the Second International Conference on Concurrent Engineering Research and Applications*, McLean, Virginia, August 1995.

Decker, K. and Lesser, V. "Designing a Family of Coordination Algorithms," *Proceedings of the First International Conference on Multiagent Systems*, San Francisco, June 1995. AAAI Press.

Decker, K.S. and Lesser, V.R. "Designing a family of coordination algorithms" in the *Proceedings of 13th International Distributed Artificial Intelligence Workshop*, July 1994.

Decker, K.S. and Lesser, V.R. "Communication in the Service of Coordination," in the *Proceedings of the Workshop on Planning for Interagent Communication*, AAAI, Seattle, WA, July 1994.

Decker, K. and Lesser, V. "Task Environment Centered Design of Organizations" in *Computational Organization Design, Working Notes of the AAAI Spring Symposium*, Ingemar Hulthage (ed.), 1994.

Decker, K. and Lesser, V., "Examples of Quantitative Modeling of Complex Computational Task Environments," Workshop on AI and Theories of Groups & Organizations: Conceptual and Empirical Research, AAAI-93, Washington, DC, 1993.

Decker, K. and Lesser, V.R. "Quantitative Modeling of Complex Environments," *International Journal of Intelligent Systems in Accounting, Finance and Management, special issue on Mathematical and Computational Models of Organizations: Models and Characteristics of Agent Behavior.* Vol. 2: 215-234, 1993.

Decker, K. and Lesser, V., "Analyzing a Quantitative Coordination Relationship," *Group Decision and Negotiation*, 2:195–217, 1993.

Decker, K. and Lesser, V. "An Approach to Analyzing the Need for Meta-Level Communication," *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, 1993.

Decker, K. and Lesser, V. "A One-Shot Dynamic Coordination Algorithm for Distributed Sensor Networks," *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pp. 210–216, 1993.

Decker, K. and Lesser, V. "Quantitative Modeling of Complex Computational Task Environments," *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pp. 217–224, 1993.

Decker, K. and Lesser, V., "Quantitative Modeling of Complex Computational Task Environments," *Twelfth International Workshop on Distributed Artificial Intelligence*, 1993.

Decker, K. and Lesser, V., "Analyzing the Need for Meta-Level Communication," *Twelfth International Workshop on Distributed Artificial Intelligence,* 1993.

Decker, K. S., Garvey, A.J., Lesser, V.R., and Humphrey, M.A., "An Approach to Modeling Environment and Task Characteristics for Coordination," *AAAI Workshop on Enterprise Integration*, San Jose, July 1992.

Decker, K. and Lesser, V., "Generalizing The Partial Global Planning Algorithm," *International Journal on Intelligent Cooperative Information Systems,* 1(2):319-346, 1992.

Garvey, A. and Lesser, V. "Design-to-time Scheduling and Anytime Algorithms." *Proceedings of the Workshop on Anytime Algorithms and Deliberation Scheduling*, IJCAI-95, Montreal, Canada.

Garvey, A. and Lesser, V., "A Survey of Research in Deliberative Real-Time Artificial Intelligence," *The Journal of Real-Time Systems*, 6(3):317-347, May 1994.

Garvey, A. and Lesser, V., "Research Summary of Investigations Into Optimal Design-to-time Scheduling," in *Proceedings of AAAI Workshop on Experimental Evaluation of Reasoning and Search Methods,* Seattle, WA, July 1994.

Garvey, A., Decker, K. and Lesser, V. "A Negotiation-based Interface Between a Real-time Scheduler and a Decision-Maker," in *Proceedings of Workshop on Models of Conflict Management in Cooperative Problem Solving*, AAAI, Seattle, WA, July 1994.

Garvey, A., Humphrey, M., and Lesser, V. "Task Interdependencies in Design-to-time Real-time Scheduling," *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pp. 580–585, 1993.

Garvey, A. and Lesser, V., "Scheduling Satisficing Tasks with a Focus on Design-to-time Scheduling," *Proceedings of IEEE Workshop on Imprecise and Approximate Computation*, Phoenix, AZ, pp. 25-29, December 1992.

Lander, S. and Lesser, V. "Organizing Cooperative Search Among Heterogeneous Expert Agents," *AI in Collaborative Design Workshop*, AAAI-93, Washington, D.C., 1993.

Lander, S. and Lesser, V. "Understanding the Role of Negotiation in Distributed Search Among Heterogeneous Agents," *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, Chambery, France, 1993.

Lander, S. and Lesser, V. "Understanding the Role of Negotiation in Distributed Search Among Heterogeneous Agents," *IJCAI-93 Workshop on Computational Models of Conflict Management in Cooperative Problem Solving*, IJCAI-93, Chambery, France, August 1993.

Lander, S.E., Lesser, V.R. and Nagendraprasad, M. "Knowledge Sharing Among Heterogeneous Reusable Agents in Cooperative Distributed Search" in the *Proceedings of 13th International Distributed Artificial Intelligence Workshop*, Seattle, July 1994.

Lander, S. and Lesser, V. "Understanding the Role of Negotiation in Distributed Search among Heterogeneous Agents," *Twelfth International Workshop on Distributed Artificial Intelligence,* 1993.

Lander, S. and Lesser, V., "Customizing Distributed Search Among Agents with Heterogeneous Knowledge," *Proceedings of the First International Conference on Information and Knowledge Management*, pp. 335-344, Baltimore, MD, November 1992.

Lander, S. and Lesser, V.R. "Negotiated Search: Organizing Cooperative Search Among Heterogeneous Expert Agents," in *Proceedings of the Fifth International Symposium on Artificial Intelligence, Applications in Manufacturing and Robotics*, Cancun, Mexico, December 1992.

Mammen, D. and Victor R. Lesser, "Using Textures to Control Distributed Problem Solving," in *Proceedings of the Workshop on Cooperation Among Heterogeneous Intelligent Systems, AAAI-92,* San Jose, July 1992.

Neiman, D., Hildum, D., Lesser, V. and Sandholm, T. "Exploiting Meta-level Information in a Distributed Scheduling System," in *Proceedings of Twelfth National Conference on Artificial Intelligence*, Seattle, WA, July/August 1994.

Oates, T., Nagendra Prasad, M.V., Lesser, V.R. , and Decker, K.S. "A Distributed Problem Solving Approach to Cooperative Information Gathering," *AAAI Spring Symposium*, Stanford CA, March 1995.

Sandholm, T. and Lesser, V. "Coalition Formation among Bounded Rational Agents." *14th International Joint Conference on Artificial Intelligence* (IJCAI-95), Montreal, Canada, August 1995.

Sandholm, T. and Lesser, V. "Equilibrium Analysis of the Possibilities of Unenforced Exchange in Multiagent Systems." *14th International Joint Conference on Artificial Intelligence* (IJCAI-95), Montreal, Canada, August 1995.

Sandholm, T. and Lesser, V. "On Automated Contracting in Multi-enterprise Manufacturing." *Proceedings of Conference on Improving Manufacturing Performance in a Distributed Enterprise: Advanced Systems and Tools*, Edinburgh, Scotland, July 1995.

Sandholm, T. and Lesser, V. "Issues in Automated Negotiation and Electronic Commerce: Extending the Contract Net Framework." in *Proceedings of First International Conference on Multiagent Systems* (ICMAS-95), San Francisco, June 1995.

Sandholm, T. "A New Order Parameter for 3SAT," in *Proceedings of the AAAI-94 Workshop on Experimental Evaluation of Reasoning and Search Methods*, August 1994.

Sandholm, T. and Lesser, V. "Utility-Based Termination of Anytime Algorithms," in the *Proceedings of European Conference on AI: 1994 Workshop on Decision Theory for DAI Applications*, 1994.

Sandholm, T. and Lesser, V.R. "An Exchange Protocol Without Enforcement" in the *Proceedings of 13th International Distributed Artificial Intelligence Workshop*, Seattle, July 1994.

Sandholm, T. "An Implementation of the Contract Net Protocol Based on Marginal Cost Calculations," *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pp. 256–262, 1993.

Sandholm, T. "An Implementation of the Contract Net Protocol Based on Marginal Cost Calculations," *Twelfth International Workshop on Distributed Artificial Intelligence*, 1993.

Sugawara, T. and Lesser, V. "Learning Coordination Plans in Distributed Problem-Solving Environments." Abstract in *Proceedings of the First International Conference on Multi-Agent Systems*, San Francisco, June 1995. AAAI Press.

Sugawara, T. and Lesser, V., "On-Line Learning of Coordination Plans," *Twelfth International Workshop on Distributed Artificial Intelligence*, 1993.

4.3 Unrefereed Reports and Articles:

Decker, K. and Lesser, V. "Coordination Assistance for Mixed Human and Computational Agents." Computer Science Technical Report 95–31, University of Massachusetts, 1995.

Decker, K. and Lesser, V.R. "Designing a Family of Coordination Algorithms," Computer Science Technical Report 94–14, University of Massachusetts at Amherst, January 1994.

Decker, K. and Lesser, V. "Quantitative Modeling of Complex Computational Task Environments," Computer Science Technical Report 93–21, University of Massachusetts, 1993.

Decker, K. and Lesser, V., "Analyzing the Need for Meta-Level Communication," Computer Science Technical Report 93–22, University of Massachusetts, 1993.

Garvey, A. and Lesser, V. "Design-to-time Scheduling With Uncertainty," University of Massachusetts Computer Science Department Technical Report 95–03, 1995.

Garvey, A., Decker, K. and Lesser, V. "A Negotiation-based Interface Between a Real-time Scheduler and a Decision-Maker," Computer Science Technical Report 94–08, University of Massachusetts at Amherst, January 1994.

Garvey, A. and Lesser, V., "A Survey of Research in Deliberative Real-Time Artificial Intelligence," Computer Science Technical Report 93–84, University of Massachusetts, November 1993.

Lander, S.E. and Lesser, V.R. "Sharing Meta-Information to Guide Cooperative Search among Heterogeneous Reusable Agents," Computer Science Technical Report 94–48, University of Massachusetts, June 1994.

Mammen, D., Fujita, S. and Lesser, V. "Predicting Solution Time for Constraint Satisfaction Problems Using Backtracking." Computer Science Technical Report 95–44, University of Massachusetts, Amherst, 1995.

Mammen, D. and Lesser, V.R., "Interdependent Suproblems in Distributed Problem Solving," Computer Science Technical Report 93-58, University of Massachusetts, 1993.

Nagendra Prasad, M., Lesser, V., and Lander, S. "Retrieval and Reasoning in Distributed Case Bases." Computer Science Technical Report 95–27, University of Massachsuetts at Amherst, March 1995.

Oates, T., Nagendra Prasad, M. and Lesser, V. "Cooperative Information Gathering: A Distributed Problem Solving Approach." Computer Science Technical Report 94–66, University of Massachusetts at Amherst, 1994.

Sandholm, T. and Lesser, V. "Advantages of a Leveled Commitment Contracting Protocol." Extended version. University of Massachusetts at Amherst, Computer Science Technical Report 95-72, 1995.

Sandholm, T. and Lesser, V. "Coalition Formation among Bounded Rational Agents." Extended version. University of Massachusetts at Amherst, Computer Science Technical Report TR 95-71, 1995.

Sandholm, T. and Lesser, V.R. "An Exchange Protocol Without Enforcement." Computer Science Technical Report 94–44, University of Massachusetts, July 1994.

Sandholm, T. and Lesser, V. "Utility-Based Termination of Anytime Algorithms." Computer Science Technical Report 94–54, University of Massachusetts, July 1994.

Sugawara, T. and Lesser, V., "On-Line Learning of Coordination Plans," Computer Science Technical Report 93–27, University of Massachusetts, 1993.

4.4 Books or Parts Thereof Published

Decker, K. S., Garvey, A.J., Lesser, V.R., and Humphrey, M.A., "An Approach to Modeling Environment and Task Characteristics for Coordination," in *Enterprise Integration Modeling*: *Proceedings of the First International Conference,* Charles J. Petrie, Jr. (ed.). Cambridge: MIT Press, 1992, pp. 379–388.

Garvey, A. and Lesser, V., "Representing and Scheduling Satisficing Tasks," in *Imprecise and Approximate Computation,* S. Natarajan (ed.). Norwell, MA: Kluwer Academic Publishers, pp. 23-34, 1995.

4.5 Books or Parts Thereof Accepted but not yet Published

Decker, K.S., "Distributed Artificial Intelligence Testbeds," to appear in *Foundations of Distributed Artificial Intelligence*, Wiley Inter- Science, Chapter 5, G. O'Hare and N. Jennings (eds.).

Decker, K.S., "TAEMS: A framework for analysis and design of coordination mechanisms," to appear in *Foundations of Distributed Artificial Intelligence*, Wiley Inter-Science, Chapter 17, G. O'Hare and N. Jennings (eds.).

4.6 Ph.D. Dissertations

Decker, Keith S. "Environment Centered Analysis and Design of Coordination Mechanisms," Ph.D. Dissertation and Computer Science Technical Report 95–69, University of Massachusetts at Amherst, May 1995.

Hildum, David W. "Flexibility in a Knowledge-Based System for Solving Dynamic Resource-Constrained Scheduling Problems," Ph.D. Dissertation and Computer Science Technical Report 94–77, University of Massachusetts at Amherst, September 1994.

Lander, Susan E. "Distributed Search and Conflict Management Among Reusable Heterogeneous Agents," Ph.D. Dissertation and Computer Science Technical Report 94–32, University of Massachusetts at Amherst, April 1994.

5 TECHNOLOGY TRANSFER

FY92

- Professor Paul Cohen is using the underlying scheduler developed for the airplane scheduler application to do the scheduling of shipping in his transportation simulation framework.

FY93

- Professor Victor Lesser's contributions to the ARPA Review Panel on Advanced Distributed Simulation, headed by Dexter Fletcher of IDA for Colonel Reddy at ARPA, was greatly influenced by the research on this contract.

FY94-95

- Collaboration with Professor Lee Osterweil (who is supported under ARPA funds in the Arcadia Project) to use their knowledge-based scheduler to schedule software activities.
- Generated proposals with Raytheon and Crystaliz to develop negotiation protocols for use with KQML.
- Continued collaboration with Professor Lee Osterweil (who is supported under ARPA funds in the Arcadia Project) to use the DSS knowledge-based scheduler to schedule software activities.
- Experiences gained from the TEAM concurrent engineering architecture is being applied to Ford Research Labs problems through a contract with Blackboard Technology, Inc.

APPENDICES

APPENDIX A

Designing a Family of Coordination Algorithms [1]

Keith S. Decker and Victor R. Lesser
Department of Computer Science
University of Massachusetts, Amherst, MA 01003
DECKER@CS.UMASS.EDU

UMass Computer Science Technical Report 94–14
August 9, 1995

Abstract

Many researchers have shown that there is no single best organization or coordination mechanism for all environments. This paper discusses the design and implementation of an extendable family of coordination mechanisms, called Generalized Partial Global Planning (GPGP). The set of coordination mechanisms described here assists in scheduling activities for teams of cooperative computational agents. The GPGP approach has several unique features. First, it is not tied to a single domain. Each mechanism is defined as a response to certain features in the current task environment. We show that different combinations of mechanisms are appropriate for different task environments. Secondly, the approach works in conjunction with an agent's existing local planner/scheduler. Finally, the initial set of five mechanisms presented here generalizes and extends the Partial Global Planning (PGP) algorithm. In comparison to PGP, GPGP schedules tasks with deadlines, it allows agent heterogeneity, it exchanges less global information, and it communicates at multiple levels of abstraction. We analyze the performance of several GPGP algorithm family members and one centralized upper bound reference algorithm, using data from simulations of multiple agent teams working in abstract task environments. We show how to decide if adding a new mechanism is useful, and suggest a way to prune the search for an appropriate combination of mechanisms in an environment.

[1]A shorter version of this paper appeared in the *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, San Francisco, June 1995. This work was supported by DARPA contract N00014-92-J-1698, Office of Naval Research contract N00014-92-J-1450, and NSF contract IRI-9321324. The content of the information does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

1 Introduction

This paper presents a formal description of the implementation of a domain independent scheduling coordination approach which we call Generalized Partial Global Planning (GPGP). The GPGP approach consists of an extendable set of modular coordination mechanisms, any subset or all of which can be used in response to a particular task environment. Each mechanism is defined using our formal framework for expressing coordination problems (TÆMS [8]). GPGP both generalizes and extends the Partial Global Planning (PGP) algorithm [10].

Our approach has several unique features:

- Each mechanism is defined as a response to certain features in the current subjective task environment. Each mechanism can be removed entirely, or can be parameterized so that it is only active for some portion of an episode. New mechanisms can be defined; an initial set of five mechanisms is examined that together approximate the original PGP behavior. Eventually we intend to develop a library of reusable coordination mechanisms. The individual coordination mechanisms rest on a shared substrate that arbitrates between the mechanisms and the agent's local scheduler in a decision-theoretic manner.

- GPGP works in conjunction with an existing agent architecture and local scheduler. The experimental results reported here were achieved using a 'design-to-time' real-time local scheduler developed by Garvey [13].

- GPGP, unlike PGP, is not tied to a single domain. GPGP allows more agent heterogeneity than PGP with respect to agent capabilities. GPGP mechanisms in general exchange less information than the PGP algorithm, and the information that GPGP mechanisms exchange can be at different levels of abstraction. PGP agents communicated complete schedules at a single, fixed level of abstraction. GPGP mechanisms communicate scheduling commitments to particular tasks, at any convenient level of abstraction.

The GPGP approach views coordination as *modulating* local control, not replacing it. This process occurs via a set of domain-independent coordination mechanisms that post constraints to the local scheduler about the importance of certain tasks and appropriate times for their initiation and completion. An example of a GPGP coordination mechanism is the one that handles simple method redundancy. If more than one agent has an otherwise equivalent method for accomplishing a task, then an agent that schedules such a method will commit to executing it, and will notify the other agents of its commitment. If more than one agent should happen to commit to a redundant method, the mechanism takes care of retracting all but one of the redundant commitments.

By concentrating on the creation of local scheduling constraints, we avoid the sequentiality of scheduling in the original PGP algorithm that occurs when there are multiple plans. By having separate modules for coordination and local scheduling, we can also take advantage of advances in real-time scheduling to produce cooperative distributed problem solving systems that respond to real-time deadlines. We can also take advantage of local schedulers that have a great deal of domain scheduling knowledge already encoded within them. Finally, our approach allows consideration of termination issues that were glossed over in the PGP work (where termination was handled by an external oracle). Nothing in TÆMS the underlying task structure representation, requires agents to be cooperative, antagonistic, or simply self-motivated.

Besides the obvious connections to the earlier PGP work, GPGP builds on work by von Martial [20] in detecting and reacting to relationships (such as von Martial's "favor" relationship). GPGP also uses a notion of social commitments similar to those discussed by [2, 19, 1, 15]. Durfee's newer work [9] is based on a hierarchical behavior space representation that like GPGP allows agents to communicate at multiple levels of detail. The mechanisms presented in this paper deal with coordination while agents are scheduling (locating in time) their activities rather than while they are planning to meet goals. This allows them to be used in distributed scheduling systems, agenda-based systems (like blackboard systems), or systems where agents instantiate previous plans (like case-based planning systems). The focus on mechanisms for coordinating schedules is thus slightly different from work that focuses on multi-agent planning [14, 11]. Shoham and Tennenholtz's 'social laws' approach [18] can be viewed as one which tries to change the (perceived) structure of the tasks by, for example, restricting the agents' possible activities. Intelligent agents might use all of these approaches at one time or another.

The next section will briefly re-introduce our framework for representing coordination problems, and summarize the assumptions we make about an agent's internal architecture. We then describe the GPGP substrate and five coordination mechanisms.[1] Previous work has shown how the GPGP approach can duplicate and extend the behaviors of the PGP algorithm [5]; Section 4 summarizes several new results that are reported in [4] concerning this approach's performance, adaptability, and extendibility. We conclude with a look at our future directions.

1.1 Representing The Task Environment

Coordination is the process of managing interdependencies between activities [17]. If we view an agent as an entity that has some beliefs about the world and can perform actions, then the coordination problem arises when any or all of the following situations occur: the agent has a *choice* of actions it can take, and that choice affects the agent's performance; the *order* in which actions are carried out affects performance; the *time* at which actions are carried out affects performance. The coordination problem of choosing and temporally ordering actions is made more complex because the agent may only have an incomplete view of the entire task structure of which its actions are a part, the task structure may be changing dynamically, and the agent may be uncertain about the outcomes of its actions. If there are multiple agents in an environment, then when the potential actions of one agent are related to those of another agent, we call the relationship a *coordination relationship.* Each GPGP coordination mechanism is a response to some coordination relationship.

The TÆMS framework (Task Analysis, Environment Modeling, and Simulation) [8] represents coordination problems in a formal, domain-independent way. We have used it to represent coordination problems in distributed sensor networks, hospital patient scheduling, airport resource management, distributed information retrieval, pilot's associate, local area network diagnosis, etc. [4]. In this paper we will describe an agent's current subjective beliefs

[1]These five mechanisms are oriented towards producing PGP-like 'cooperative team' behavior. Mechanisms for self-interested agents are also possible.

about the structure of the problem it is trying to solve by using the TÆMS framework [8, 4]. For this purpose, there are two unique features of TÆMS. The first is the explicit, quantitative representation of task interrelationships as functions that describe the effect of activity choices and temporal orderings on performance. The second is the representation of task structures at multiple levels of abstraction. The highest level of abstraction is called a *task group*, and contains all tasks that have explicit computational interrelationships. A *task* is simply a set of lower-level subtasks and/or executable methods. The components of a task have an explicitly defined effect on the quality of the encompassing task. The lowest level of abstraction is called an executable *method.* An executable *method* represents a schedulable entity, such as a blackboard knowledge source instance, a chunk of code and its input data, or a totally-ordered plan that has been recalled and instantiated for a task. A method could also be an instance of a human activity at some useful level of detail, for example, "take an X-ray of patient 1's left foot".

A coordination problem instance (called an *episode* $\mathbf{E}$) is defined as a set of task groups, each with a deadline $\mathrm{D}(\mathcal{T})$, such as $\mathbf{E} = \langle \mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n \rangle$. Figure 1 shows an objective[2] task group and agent A's subjective view of that same task group. A common performance goal of the agent or agents is to maximize the sum of the quality achieved for each task group before its deadline. A task group consists of a set of tasks related to one another by a subtask relationship that forms an acyclic graph (here, a tree). Tasks at the leaves of the tree represent executable *methods*, which are the actual instantiated computations or actions the agent will execute that produce some amount of quality (in the figure, these are shown as boxes). The circles higher up in the tree represent various subtasks involved in the task group, and indicate precisely how quality will accrue depending on what methods are executed and when. The arrows between tasks and/or methods indicate other task interrelationships where the execution of some method will have a positive or negative effect on the quality or duration of another method. The presence of these interrelationships make this an NP-hard scheduling problem; further complicating factors for the local scheduler include the fact that multiple agents are executing related methods, that some methods are redundant (executable at more than one agent), and that the subjective task structure may differ from the real objective structure.

2 Summary of the GPGP algorithm family approach

This section will provide a quick overview of the GPGP approach. Figure 2 shows a simple two-agent example that we will use. Each agent has as part of its architecture a belief database, local scheduler, and coordination module. The local scheduler uses the information in the belief database to schedule method execution actions for the agent in an attempt to maximize its performance. We add to this a coordination module that is in charge of communication actions, information gathering actions, and in making and breaking *commitments* to complete tasks in the task structure. The coordination module consists of several coordination mechanisms, each of which notices certain features in the task structures in the belief database, and responds by taking certain communication or information gathering actions, or by proposing new commitments. The coordination mechanisms rest in a shared coordination module substrate that keeps track of local commitments and commitments received from other agents, and that chooses from among multiple schedules if the local scheduler returns multiple schedules.

[2]The word 'objective' refers to the fact that this is the true, real structure.

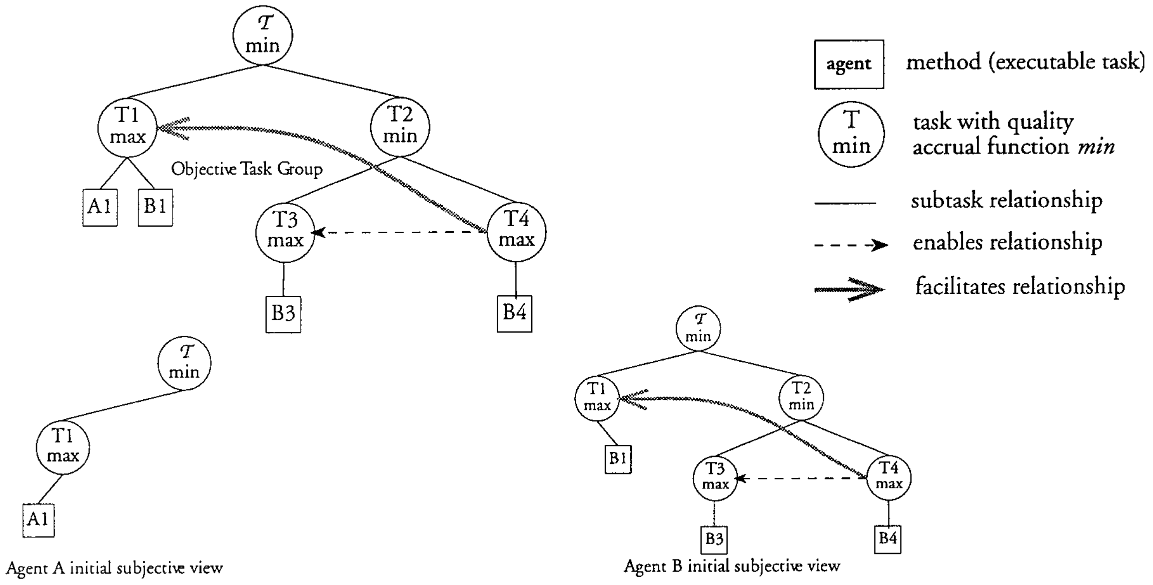

Figure 1: Agent A and B's subjective views (bottom) of a typical objective task group (top)

In these environments, the agents attempt to maximize the system-wide total utility (a quantity called 'quality', described later) by executing sequences of interrelated 'methods'. The agents do not initially have a complete view of the problem solving situation, and the execution of a method at one agent can either positively or negatively affect the execution of other methods at other agents. We will show examples of the effect of the environment on the performance of a GPGP family member, and show an environment where family member A is better than B, and a different environment where B is better than A. We will return to the demonstration of meta-level information being more useful when there is a large amount of variance between episodes in an environment.

Here is a short example intended only to give the reader a feel for the overall approach. In Figure 2, both agents have executed an initial information gathering action, and have their initial views of the task structure (everything in the agents' belief database *except* for the shaded tasks (Tasks 2, 5, D and E), and the relationships touching the shaded tasks). One of the coordination mechanisms (Mech. 1, update non-local views) performs an information gathering action to determine which tasks may be related to tasks at other agents ("detect coordination relationships"). These tasks are then exchanged between the agents, resulting in the belief databases shown in the figure (including the shaded tasks). Other mechanisms react to the task structure. One mechanism (Mech. 5, handle soft predecessors) notices that Task 2 at Agent Y **facilitates** Task 5 at Agent X. In order that Agent X might schedule to take advantage of this, Agent Y's mechanism makes a local intermediate deadline commitment to complete its Task 2 by time 7 with minimum quality 45 (you and I may infer that Y intends to execute Method B, but that local information is not a part of the commitment). A commitment is made in two stages: first it is made locally to see if it is possible as far as the agent's local scheduler is concerned, and then it is made non-locally and communicated to the other agents that are involved. Note that the deadline on the non-local version of this commitment is later (time 8) to take into account the communication delay (here, 1 time unit). Similarly, Agent

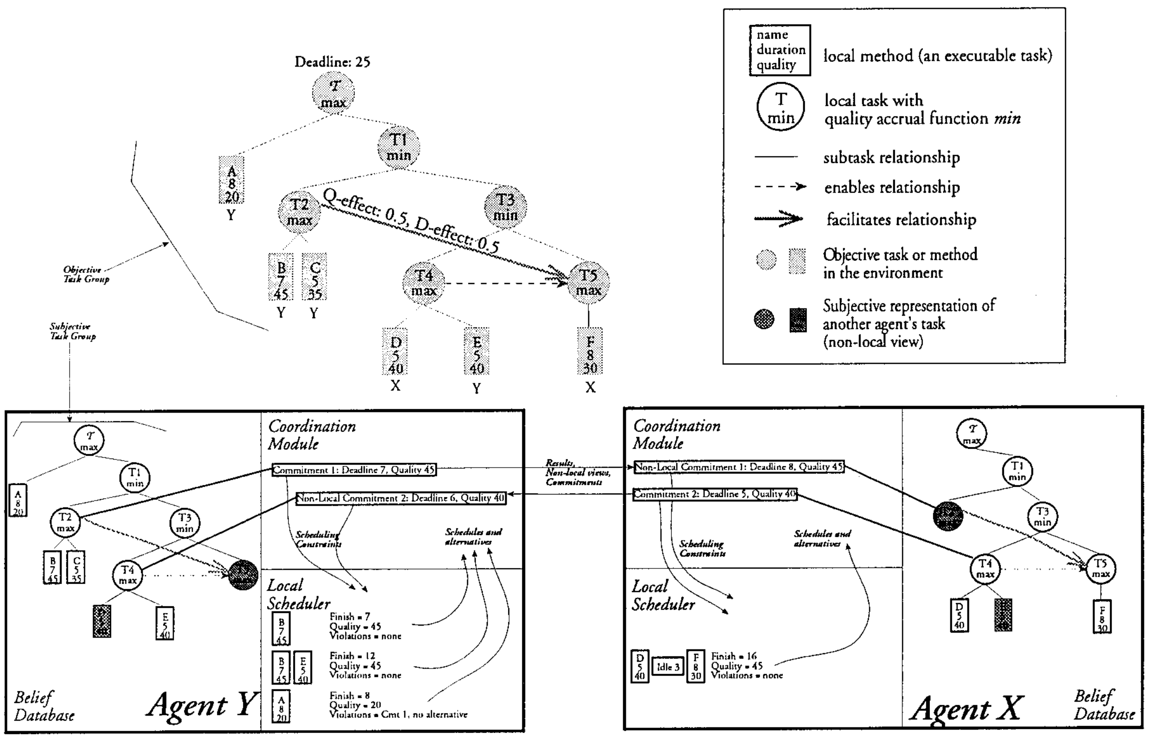
Deadline: 25
name
duration
quality
local method (an executable task)
local task with quality accrual function *min*
subtask relationship
enables relationship
facilitates relationship
Objective task or method in the environment
Subjective representation of another agent's task (non-local view)
Objective Task Group
Subjective Task Group
Q-effect: 0.5, D-effect: 0.5
Coordination Module
Commitment 1: Deadline 7, Quality 45
Non-Local Commitment 2: Deadline 6, Quality 40
Scheduling Constraints
Schedules and alternatives
Local Scheduler
Finish = 7
Quality = 45
Violations = none
Finish = 12
Quality = 45
Violations = none
Finish = 8
Quality = 20
Violations = Cmt 1, no alternative
Belief Database
Agent Y
Results, Non-local views, Commitments
Non-Local Commitment 1: Deadline 8, Quality 45
Commitment 2: Deadline 5, Quality 40
Idle 3
Finish = 16
Quality = 45
Violations = none
Agent X

Figure 2: An Overview of Generalized Partial Global Planning

X has a mechanism (Mech. 3, handle simple redundancy) that notices that either agents X or Y could do Task 4. Agent X does eventually commit to this task (the process is a bit more complicated as will be explained later) and communicates this commitment to Agent Y.

In both cases the agents' local schedulers use the information about the task structure they have in their belief database, and the local and non-local commitments, to construct schedules. The local scheduler may return multiple schedules for several reasons we explain later. Each schedule is evaluated along the dimensions of the performance criteria (such as total final quality and termination time) and for what (if any) local commitments are violated. If a commitment is violated, the local scheduler may suggest an alternative (for instance, relaxing a quality or intermediate deadline constraint). The coordination module chooses a schedule from this set, and handles the retraction of any violated commitments.

2.1 The Agent Architecture

We make few assumptions about the architecture of the agents. The agents have a database that holds their current beliefs about the structure of the tasks in the current episode; we represent this information using TÆMS. The agents can do three types of actions: they can execute methods from the task structure, send direct messages to one another, and do "information gathering". Information gathering actions model how new task structures or communications get into the agent's belief database. This could be a combination of external actions (checking the agent's incoming message box) and internal planning. Method execution actions cause quality to accrue in a task group (as indicated by the task structure). Communication actions are used to send the results of method executions (which in turn may trigger the effects of various task interrelationships) or meta-level information.

Formally, we write $B_A^t(x)$ to mean agent A subjectively believes x at time t (from Shoham[19]). We will shorten this to $B(x)$ when the particular agent or time is not important. An agent's subjective beliefs about the current episode include the agent's beliefs about task groups, subtasks, executable methods, and interrelationships (e.g., $B(\mathcal{T}_i \in \mathbf{E}), B(T_a, M_b \in \mathcal{T}_i), B(\mathsf{enables}(T_a, M_b)))$.

The GPGP family of coordination mechanisms also makes a stronger assumption about the agent architecture. It assumes the presence of a local scheduling mechanism (to be described in the next section) that can decide what method execution actions should take place and when. The local scheduler attempts to maximize a (possibly changing) utility function. The current set of GPGP coordination mechanisms are for cooperative teams of agents—they assume that agents do not intentionally lie and that agents believe what they are told. However, because agents can believe and communicate only subjective information, they may unwittingly transmit information that is inconsistent with an objective view (this can cause, among other things, the phenomena of *distraction*). Finally, the GPGP family approach requires domain-dependent code to detect or predict the presence of coordination relationships in the local task structure. In this paper we will refer to that domain-dependent code as the information gathering action called *detect-coordination-relationships*; we will describe this action more in Section 3.2.

2.2 The Local Scheduler

Each GPGP agent contains a local scheduler that takes three types of input information and produces a set of schedules and alternatives. The first input is the current, subjectively believed task structure. Using information about the potential duration, potential quality, and interrelationships, the local scheduler chooses and orders executable methods in an attempt to maximize a pre-defined utility function. In this paper the utility function is the sum of the task group qualities $\sum_{\mathcal{T} \in \mathbf{E}} Q(\mathcal{T}, \mathrm{D}(\mathcal{T}))$, where $Q(T,t)$ denotes the quality of T at time t as defined in [8]. Quality does not accrue after a task group's deadline.

The second input is a set of *commitments* $\mathbf{C}$. These commitments are produced by the GPGP coordination mechanisms, and act as extra constraints on the schedules that are produced by the local scheduler. For example, if method 1 is executable by agent A and method 2 is executable by agent B, and the methods are redundant, then one of agent A's coordination mechanisms may *commit* agent A to do method 1. Commitments are *social*—directed to particular agents in the sense of the work of Shoham and Castelfranchi [1, 19]). A local commitment C by agent A becomes a non-local commitment when received by another agent B. This paper will use two types of commitments: $C(\mathsf{Do}(T, q))$ is a commitment to 'do' (achieve quality for) T and is satisfied at the time t when $Q(T,t) \geq q$; the second type $C(\mathsf{DL}(T, q, t_{dl}))$ is a 'deadline' commitment to do T by time t_{dl} and is satisfied at the time t when $[Q(T,t) \geq q] \wedge [t \leq t_{dl}]$. When a commitment is sent to another agent, it also implies that the task result will be communicated to the other agent (by the deadline, if it is a deadline commitment).

The third input to the local scheduler is the set of non-local commitments $\mathbf{NLC}$ made by other agents. This information can be used by the local scheduler to coordinate actions between agents. For example the local scheduler could have the property that, if method M_1 is executable by agent A and is the only method that **enables** method M_2 at agent B (and agent B knows this), and $B_A(C(\mathsf{DL}(M_1, q, t_1))) \in B_B(\mathbf{NLC})$, then for every schedule S produced by agent B, $\langle M_2, t\rangle \in S \;\Rightarrow\; t \geq t_1$ (in other words, agent B only schedules the enabled method after the deadline that agent A has committed to.

A schedule S produced by a local scheduler will consist of a set of methods and start times: $S = \{\langle M_1, t_1\rangle, \langle M_2, t_2\rangle, \ldots, \langle M_n, t_n\rangle\}$. The schedule may include idle time, and the local scheduler may produce more than one schedule upon each invocation in the situation where not all commitments can be met. The different schedules represent different ways of partially satisfying the set of commitments. The function Violated(S) returns the set of commitments that are believed to be violated by the schedule. For violated deadline commitments $C(\mathsf{DL}(T, q, t_{dl})) \in$ Violated(S) the function Alt(C, S) returns an alternative commitment $C(\mathsf{DL}(T, q, t^*_{dl}))$ where $t^*_{dl} = \min t$ such that $Q(T, t) \geq q$ if such a t exists, or NIL otherwise. For a violated Do commitment an alternative may contain a lower minimum quality, or no alternative may be possible. The function $U_{\mathbf{est}}(\mathbf{E}, S, \mathbf{NLC})$ returns the estimated utility at the end of the episode if the agent follows schedule S and all non-local commitments in $\mathbf{NLC}$ are kept.

Thus we may define the local scheduler as a function $\mathrm{LS}(\mathbf{E}, \mathbf{C}, \mathbf{NLC})$ returning a set of schedules $\mathbf{S} = \{S_1, S_2, \ldots, S_m\}$. More detailed information about this kind of interface between the local scheduler and the coordination component may be found in [12]. This is an extremely general definition of the local scheduler, and is the minimal one necessary for the GPGP coordination module. Stronger definitions than this will be needed for more predictable performance, as we will discuss later. Ideally, the optimal local scheduler would find both the schedule with maximum utility $\hat{S_U}$ and the schedule with maximum utility that violates no commitments $\hat{S_{\bar{V}}}$. In practice, however, a heuristic local scheduler will produce a set of schedules where the schedule of highest utility S_U is not necessarily optimal: $\mathrm{U}(\mathbf{E}, S_U, \mathbf{NLC}) \leq \mathrm{U}(\mathbf{E}, \hat{S_U}, \mathbf{NLC})$.

3 Five GPGP Coordination Mechanisms

The role of the coordination mechanisms is to provide information to the local scheduler that allows the local scheduler to construct better schedules. This information can be in the form of modifications to portions of the subjective task structure of the episode or in the form of local and non-local commitments to tasks in the task structure. The five mechanisms we will describe in this paper form a basic set that provides similar functionality to the original Partial Global Planning algorithm as shown in [5]. Mechanism 1 exchanges useful private views of task structures; Mechanism 2 communicates results; Mechanism 3 handles redundant methods; Mechanisms 4 and 5 handle hard and soft coordination relationships. More mechanisms can be added, such as one to update utilities across agents as discussed in the next section, or to balance the load better between agents. The mechanisms are independent in the sense that they can be used in any combination. If inconsistent constraints are introduced, the local scheduler will return at least one violated constraint in all its schedules. Since the local scheduler typically satisfices instead of optimizes, it may do this even if constraints are not inconsistent (i.e. it does not search exhaustively). The next section describes how a schedule is chosen by the coordination module substrate.

3.1 The GPGP Coordination Module Substrate

All the specific coordination mechanisms rest on a common substrate that handles information gathering actions, invoking the local scheduler, choosing a schedule to execute (including dealing with violated or inconsistent commitments), and deciding when to terminate processing on a task group. Information gathering actions include noticing new task group arrivals and receiving communications from other agents. Information gathering is done at the start of problem solving, when communications are expected from other agents, and when the agent is otherwise idle. Communications are expected in response to certain events (such as after the arrival of a new task group) or as indicated in the set of non-local commitments **NLC**. This is the minimal general information gathering policy. Termination of processing on a task group occurs for an agent when the agent is idle, has no expected communications, and no outstanding commitments for the task group.

Choosing a schedule is more complicated. The agent's local scheduler may return multiple schedules because it cannot find a single schedule that both maximizes utility and meets all commitments. From the set of schedules **S** returned by the local scheduler, two particular schedules are identified: the schedule with the highest utility S_U and the best committed schedule S_C. If they are the same, then that schedule is chosen. Otherwise, we examine the sum of the changes in utility for each commitment. Each commitment, when created, is assigned the estimated utility $U_{\mathbf{est}}$ for the task group of which it is a part. This utility may be updated over time (when other agents depend on the commitment, for example). We then choose the schedule with the largest positive change in utility. This allows us to abandon commitments if doing so will result in higher overall utility. The coordination substrate does not use the local scheduler's utility estimate $U_{\mathbf{est}}$ directly on the entire schedule because it is based only on a local view. The coordination substrate may receive non-local information that places a higher utility on a commitment than it has locally.

For example, at time t agent A may make a commitment C_1 on task $T \in \mathcal{T}_1 \in \mathbf{E}$ that results in a schedule S_1. C_1 initially acquires the estimated utility of the task group of which it is a part, $U(C_1) \leftarrow U_{\mathbf{est}}(\{\mathcal{T}_1\}, S_1, B_A(\mathbf{NLC}))$. Let $U(C_1) = 50$. After communicating this commitment to agent B (making it part of $B_B(\mathbf{NLC})$, agent B uses the commitment to improve $U_{\mathbf{est}}(\{\mathcal{T}_1\}, S_2, B_B(\mathbf{NLC}))$ to 100. A coordination mechanism can detect this discrepancy and communicate the utility increase back to agent A, so that when agent A considers discarding the commitment, the coordination substrate recognizes the non-local utility of the commitment is greater than the local utility.

If both schedules have the same utility, the one that is more negotiable is chosen. Every commitment has a negotiability index (high, medium, or low) that indicates (heuristically) the difficulty in rescheduling if the commitment is broken. This index is set by the individual coordination mechanisms. For example, hard coordination relationships like enables that cannot be ignored will trigger commitments with low negotiability. If the schedules are still equivalent, the shorter one is chosen, and if they are the same length, one is chosen at random.

After a schedule S is chosen, if Violated(S) is not empty, then each commitment $C \in$ Violated(S) is replaced with its alternative $\mathbf{C} \leftarrow \mathbf{C} \setminus C \cup \mathrm{Alt}(C, S)$. If the commitment was made to other agents, the other agents are also informed of the change in the commitment. While this could potentially cause cascading changes in the schedules of multiple agents, it generally does not for three reasons: first, as we mentioned in the previous paragraph less important commitments are broken first; secondly, the resiliancy of the local schedulers to solve problems in multiple ways tends to damp out these fluctuations; and third, agents are time cognizant resource-bounded reasoners that interleave execution and scheduling (i.e., the agents cannot spend all day arguing over scheduling details and still meet their deadlines). We have observed this useful phenomenon before [4] and plan to analyze it in future work.

3.2 Mechanism 1: Updating Non-Local Viewpoints

Remember that each agent has only a partial, subjective view of the current episode. The GPGP mechanism described here can communicate no private information ('none' policy, no non-local view), or all of it ('all' policy, global view), or take an intermediate approach ('some' policy, partial view). The process of detecting coordination relationships between private and shared parts of a task structure is in general very domain specific, so we model this process by a new information gathering action, *detect-coordination-relationships*, that takes some fixed amount of the agent's time. This action is scheduled whenever a new task group arrives.

The set $\mathbf{P}$ of privately believed tasks or methods at an agent A (tasks believed at arrival time by A only) is then $\{x \mid \mathit{task}(x) \wedge \forall a \in \mathbf{A} \setminus A,\ \neg B_A(B_a^{\mathrm{Ar}(x)}(x))\}$, where $\mathbf{A}$ is the set of all agents and $\mathrm{Ar}(x)$ is the arrival time of x. Given this definition, the action *detect-coordination-relationships* returns the set of private coordination relationships $\mathbf{PCR} = \{r \mid T_1 \in \mathbf{P} \wedge T_2 \notin \mathbf{P} \wedge [r(T_1, T_2) \vee r(T_2, T_1)]\}$ between private and mutually believed tasks. The action does not return what the task T_2 is, just that a relationship exists between T_1 and some otherwise unknown task T_2. For example, in the DVMT, we have used the physical organization of agents to detect that Agent A's task T_1 in an overlapping sensor area is in fact related to some unknown task T_2 at agent B (i.e. $B_A(B_B(T_2)))$ [5]. The non-local view coordination mechanism then communicates these coordination relationships, the private tasks, and their context: if $r(T_1, T_2) \in \mathbf{PCR}$ and $T_1 \in \mathbf{P}$ then r and T_1 will be communicated by agent A to the set of agents $\{a \mid B_A(B_a(T_2))\}$.

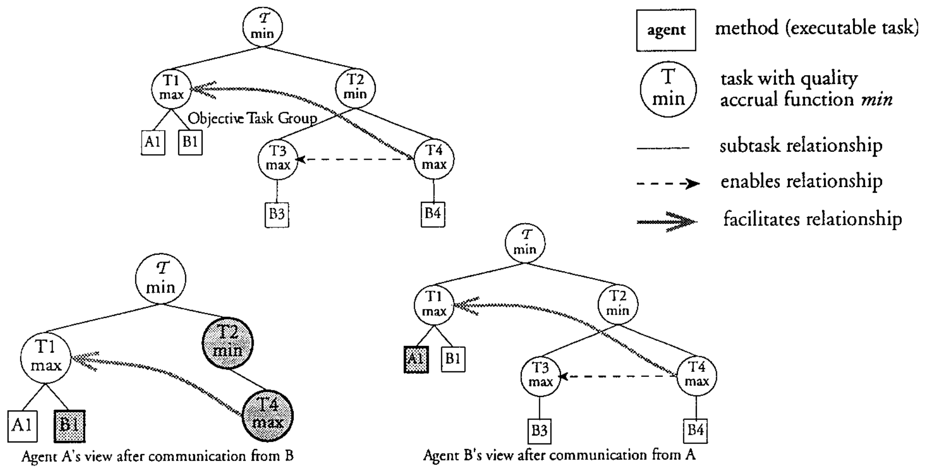

Figure 3: Agents A and B's local views after receiving non-local viewpoint communications via mechanism 1 (shaded objects). Figure 1 shows the agents' initial states.

For example, Figure 3 shows the local subjective beliefs of agents A and B after the communication from one another due to this mechanism.

The agents' initial local view was shown previously in Figure 1. In this example, T_3 and T_4 are two elements in Agent B's private set of tasks $\mathbf{P}$, $\mathsf{facilitates}(T_4, T_1, \phi_d, \phi_q) \in \mathbf{PCR}$ (the facilitation relates a private task to a mutually believed task), and $\mathsf{enables}(T_4, T_3)$ is completely local to Agent B (it relates two private tasks). At the start of this section we mentioned that coordination relationships exist **between** portions of the task structure controllable by different agents (i.e., in **PCR**) and **within** portions controllable by multiple agents. We'll denote the complete set of coordination relationships as **CR**; this includes all the elements of **PCR** and all the relationships between non-private tasks. Some relationships are entirely local—between private tasks—and are only of concern to the local scheduler. The purpose of this coordination mechanism is the exchange of information that expands the set of coordination relationships **CR**. Without this mechanism in place, **CR** will consist of only non-private relationships, and none that are in **PCR**. Since the primary focus of the coordination mechanisms is the creation of social commitments in response to coordination relationships (elements of **CR**), this mechanism can have significant indirect benefits. In environments where $|\mathbf{PCR}|$ tends to be small, very expensive to compute, or not useful for making commitments (see the later sections), this mechanism can be sucessfully omitted.

3.3 Mechanism 2: Communicating Results

The result communication coordination mechanism has three possible policies: communicate only the results necessary to satisfy commitments to other agents (the minimal policy); communicate this information plus the final results associated with a task group ('TG' policy), and communicate all results ('all' policy[3]). Extra result communications are broadcast to all agents, the minimal commitment-satisfying communications are sent only to those agents to whom the commitment was made (i.e., communicate the result of T to the set of agents $\{A \in \mathbf{A} \mid B(B_A(C(T))\}$.

3.4 Mechanism 3: Handling Simple Redundancy

Potential redundancy in the efforts of multiple agents can occur in several places in a task structure. Any task that uses a 'max' quality accumulation function (one possible semantics for an 'OR' node) indicates that, in the absence of other relationships, only one subtask needs to be done. When such subtasks are complex and involve many agents, the coordination of these agents to avoid redundant processing can also be complex; we will not address the general redundancy avoidance problem in this paper (see instead [16]). In the original PGP algorithm and domain (distributed sensor interpretation), the primary form of potential redundancy was simple method redundancy—the same result could be derived from the data from any of a number of sensors. The coordination mechanism described here is meant to address this simpler form of potential redundancy.

The idea behind the simple redundancy coordination mechanism is that when more than one agent wants to execute a redundant method, one agent is randomly chosen to execute it and send the results to the other interested agents. This is a generalization of the 'static' organization algorithm discussed by Decker and Lesser [6]—it does not try to load balance, and uses one communication action (because in the general case the agents do not know beforehand,

[3]Such a policy is all that is needed in many simple environments.

without communication, that certain methods are redundant[4]). The mechanism considers the set of potential redundancies $\mathbf{RCR} = \{r \in \mathbf{CR} \mid [r = \mathsf{subtask}(T, \mathbf{M}, \min)] \wedge [\forall M \in \mathbf{M}, \mathit{method}(M)]\}$. Then for all methods in the current schedule S at time t, if the method is potentially redundant then commit to it and send the commitment to $\mathsf{Others}(\mathbf{M})$ (non-local agents who also have a method in $\mathbf{M}$):

$$[\langle M, t_M \rangle \in S] \wedge [\mathsf{subtask}(T, \mathbf{M}, \min) \in \mathbf{RCR}] \wedge [M \in \mathbf{M}] \Rightarrow$$
$$[C(\mathsf{Do}(M, \mathsf{Q}_{\mathsf{est}}(M, \mathrm{D}(M), S))) \in \mathbf{C}] \wedge [\mathsf{comm}(M, \mathsf{Others}(\mathbf{M}), t) \in \mathcal{I}]$$

See for example the top of figure 4—both agents commit to Do their methods for T_1.

After the commitment is made, the agent must refrain from executing the method in question if possible until any non-local commitments that were made simultaneously can arrive (the communication delay time δ). This mechanism then watches for multiple commitments in the redundant set and if they appear, a unique agent is chosen randomly (but identically by all agents) from those with the best commitments to keep its commitment. All the other agents can retract their commitments. For example the bottom of figure 4 shows the situation after Agent B has retracted its commitment to Do B_1. If all agents follow the same algorithm, and communication channels are assumed to be reliable, then no second message (retraction) actually needs to be sent (because they all choose the same agent to do the redundant method). In the implementation described later, identical random choices are made by giving each method a unique random identifier, and then all agents choose the method with the 'smallest' identifier for execution.

Initially, all Do commitments initiated by the redundant coordination mechanism are marked highly negotiable. When a redundant commitment is discovered, the negotiability of the remaining commitment is lowered to medium to indicate the commitment is somewhat more important.

3.5 Mechanism 4: Handling Hard Coordination Relationships

Hard coordination relationships include relationships like $\mathsf{enables}(M_1, M_2)$ that indicate that M_1 must be executed before M_2 in order to obtain quality for M_2. Like redundant methods, hard coordination relationships can be culled from the set $\mathbf{CR}$. The hard coordination mechanism further distinguishes the direction of the relationship—the current implementation only creates commitments on the predecessors of the enables relationship. We'll let $\mathbf{HPCR} \subset \mathbf{CR}$ indicate the set of potential hard predecessor coordination relationships. The hard coordination mechanism then looks for situations where the current schedule S at time t will produce quality for a predecessor in $\mathbf{HPCR}$, and commits to its execution by a certain deadline both locally and socially:

$$[\mathsf{Q}_{\mathsf{est}}(T, \mathrm{D}(T), S) > 0] \wedge [\mathsf{enables}(T, \mathbf{M}) \in \mathbf{HPCR}] \Rightarrow$$
$$[C(\mathsf{DL}(T, \mathsf{Q}_{\mathsf{est}}(T, \mathrm{D}(T), S), t_{\mathbf{early}})) \in \mathbf{C}] \wedge [\mathsf{comm}(C, \mathsf{Others}(\mathbf{M}), t) \in \mathcal{I}]$$

The next question is, by what time ($t_{\mathbf{early}}$ above) do we commit to providing the answer? One solution, usable with any local scheduler that fits our general description in Section 2.2,

[4]The detection of redundant methods is domain-dependent, as discussed earlier. Since we are talking here about simple, direct redundancy (i.e. doing the exact same method at more than one agent) this detection is very straight-forward.

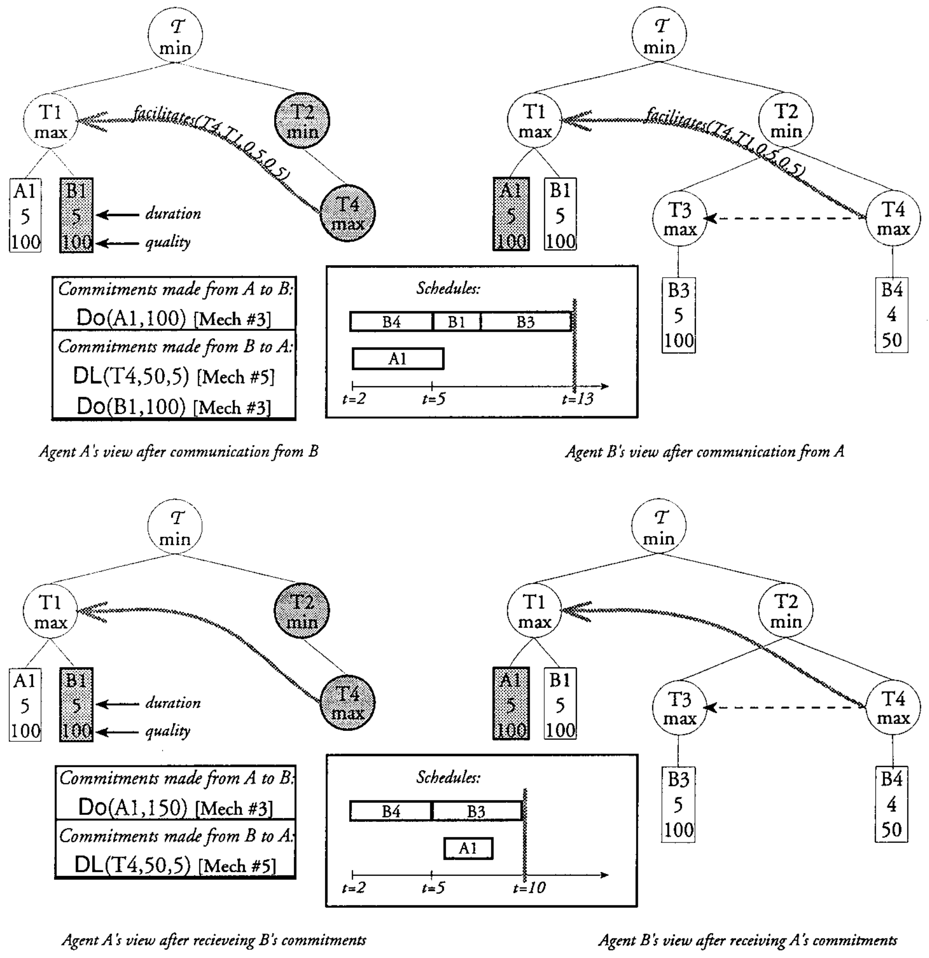
T min
T1 max
T2 min
T3 max
T4 max
facilitates(T4,T1,0.5,0.5)
A1 5 100
B1 5 100
B3 5 100
B4 4 50
duration
quality
Commitments made from A to B:
Do(A1,100) [Mech #3]
Commitments made from B to A:
DL(T4,50,5) [Mech #5]
Do(B1,100) [Mech #3]
Schedules:
B4 B1 B3
A1
t=2 t=5 t=13
Agent A's view after communication from B
Agent B's view after communication from A
Commitments made from A to B:
Do(A1,150) [Mech #3]
Commitments made from B to A:
DL(T4,50,5) [Mech #5]
Schedules:
B4 B3
A1
t=2 t=5 t=10
Agent A's view after recieveing B's commitments
Agent B's view after receiving A's commitments

Figure 4: A continuation of Figures 1 and 2. At top: agents A and B propose certain commitments to one another via mechanisms 3 and 5. At bottom: after receiving the initial commitments, mechanism 3 removes agent B's redundant commitment.

is to use the $\min t$ such that $\mathrm{Q}_{\mathrm{est}}(T, \mathrm{D}(T), S) > 0$. In our implementation, the local scheduler provides a query facility that allows us to propose a commitment to satisfy as 'early' as possible (thus allowing the agent on the other end of the relationship more slack). We take advantage of this ability in the hard coordination mechanism by adding the new commitment $C(\mathsf{DL}(T, \mathrm{Q}_{\mathrm{est}}(T, \mathrm{D}(T), S), \text{"early"}))$ to the local commitment set $\mathbf{C}$, and invoking the local scheduler $LS(\mathbf{E}, \mathbf{C}, \mathbf{NLC})$ to produce a new set of schedules $\mathbf{S}$. If the preferred, highest utility schedule $S_U \in \mathbf{S}$ has no violations (highly likely since the local scheduler can simply return the same schedule if no better one can be found), we replace the current schedule with it and use the new schedule, with a potentially earlier finish time for T, to provide a value for $t_{\mathbf{early}}$. The new completed commitment is entered locally (with low negotiability) and sent to the subset of interested other agents.

If redundant commitments are made to the same task, the earliest commitment made by any agent is kept, then the agent committing to the highest quality, and any remaining ties are broken by the same method as before.

Currently, the hard coordination mechanism is a pro-active mechanism, providing information that might be used by other agents to them, while not putting the individual agent to any extra effort. Other future coordination mechanisms might be added to the family that are reactive and request from other agents that certain tasks be done by certain times; this is quite different behavior that would need to be analyzed separately.

3.6 Mechanism 5: Handling Soft Coordination Relationships

Soft coordination relationships are handled analogously to hard coordination relationships except that they start out with high negotiability. In the current implementation the predecessor of a **facilitates** relationship is the only one that triggers commitments across agents, although **hinders** relationships are present. The positive relationship **facilitates**(M_1, M_2, ϕ_d, ϕ_q) indicates that executing M_1 before M_2 decreases the duration of M_2 by a 'power' factor related to ϕ_d and increases the maximum quality possible by a 'power' factor related to ϕ_q (see [8] for the details). A more situation-specific version of this coordination mechanism might ignore relationships with very low 'power'. The relationship **hinders**(M_1, M_2, ϕ_d, ϕ_q) is negative and indicates an *increase* in the duration of M_2 and a *decrease* in maximum possible quality. A coordination mechanism could be designed for **hinders** (and similar negative relationships) and added to the family. To be pro-active like the existing mechanisms, a **hinders** mechanism would work from the *successors* of the relationship, try to schedule them late, and commit to an earliest start time on the successor. Figure 4 shows Agent B making a D commitment to do method B_4, which in turn allows Agent A to take advantage of the **facilitates**($T_4, T1, 0.5, 0.5$) relationship, causing method A_1 to take only half the time and produce 1.5 times the quality.

4 Experimental Results

We do not believe that any of the mechanisms that collectively form the GPGP family of coordination algorithms are indispensable. What we can do is evaluate the mechanisms on the terms of their costs and benefits to cooperative problem solving both analytically and experimentally. This analysis and experimentation takes place with respect to a very general task environment that does not correspond to a particular domain. Doing this produces general results, but weaker than would be possible to derive in a single fixed domain because the performance variance between problem episodes will be far greater than the performance variance of the different algorithms within a single episode. Still, this allows us to determine broad characteristics of the algorithm family that can be used to reduce the search for a particular set of mechanism parameters for a particular domain (with or without machine learning techniques; see Section 5). We will also discuss statistical techniques (e.g. paired-response simulations) to deal with the large between-episode variances that occur when using randomly-generated problems.

4.1 GPGP Simulation: Issues

Our model of an abstract task environment, used in these experiments, has ten parameters; Table 1 lists them and the values used in the experiments described in the next two sections.[5]

[5]Our earlier work focussed on the analysis of distributed sensor network task environments [6, 7].

Figure 2 shows a small example task group.

Parameter	Values (facilitation exps.)	Values (clustering exps.)
Mean Branching factor (Poisson)	1	1
Mean Depth (Poisson)	3	3
Mean Duration (exponential)	10	(1 10 100)
Redundant Method QAF	Max	Max
Number of task groups	2	(1 5 10)
Task QAF distribution	(20%/80% min/max)	(50%/50% min/max)
		(100%/0% min/max)
Hard CR distribution	(10%/90% enables/none)	(0%/100% enables/none)
		(50%/50% enables/none)
Soft CR distribution	(80%/10%/10% facilitates/hinders/none)	(0%/10%/90% facilitates/hinders/none)
		(50%/10%/40% facilitates/hinders/none)
Chance of overlaps (binomial)	10%	(0% 50% 100%)
Facilitation Strength	.1 .5 .9	.5

Table 1: Environmental Parameters used to generate the random episodes

The primary sources of overhead associated with the coordination mechanisms include action executions (communication and information gathering), calls to the local scheduler, and any algorithmic overhead associated with the mechanism itself. Table 2 summarizes the total amount of overhead from each source for each coordination mechanism setting and the coordination substrate. L represents the length of processing (time before termination), and d is a general density measure of coordination relationships. We believe that all of these amounts can be derived from the environmental parameters in Table 1, they can also be measured experimentally. Interactions between the presence of coordination mechanisms and these quantities include: the number of methods or tasks in $\mathbf{E}$, which depends on the non-local view mechanism; the number of coordination relationships $|\mathbf{CR}|$ or the subsets $\mathbf{RCR}$ (redundant coordination relationships), $\mathbf{HPCR}$ (hard predecessor coordination relationships), $\mathbf{SPCR}$ (soft predecessor coordination relationships), which depends on the number of tasks and methods as well; and the number of commitments $|\mathbf{C}|$, which depends on each of the three mechanisms that makes commitments.

Mechanism *setting*	**Communications**	**Information Gathering**	**Scheduler**	*Other Overhead*
substrate	0	$\mathbf{E}$+*idle*	L	$O(L\mathbf{C})$
nlv *none*	0	0	0	0
some	$O(d\mathbf{P})$	$\mathbf{E}$*detect-CRs*	0	$O(T \in \mathbf{E})$
all	$O(\mathbf{P})$	$\mathbf{E}$*detect-CRs*	0	$O(T \in \mathbf{E})$
comm *min*	$O(\mathbf{C})$	0	0	$O(\mathbf{C})$
TG	$O(\mathbf{C}+\mathbf{E})$	0	0	$O(\mathbf{C}+\mathbf{E})$
all	$O(M \in \mathbf{E})$	0	0	$O(M \in \mathbf{E})$
redundant *on*	$O(\mathbf{RCR})$	0	0	$O(\mathbf{RCR} * S + \mathbf{CR})$
hard *on*	$O(\mathbf{HPCR})$	0	$O(\mathbf{HPCR})$	$O(\mathbf{HPCR} * S + \mathbf{CR})$
soft *on*	$O(\mathbf{SPCR})$	0	$O(\mathbf{SPCR})$	$O(\mathbf{SPCR} * S + \mathbf{CR})$

Table 2: Overhead associated with individual mechanisms at each parameter setting

4.2 General Performance Issues

We examined the general performance of the most complex (all mechanisms in place) and least complex (all mechanisms off) members of the GPGP family in comparison to each other, and in comparison to a centralized scheduler reference implementation (as an upper bound). We looked at performance measures such as the total final quality achieved by the system, the amount of work done, the number of deadlines missed, and the termination time. The centralized schedule reference system is not an appropriate solution to the general coordination problem, even for cooperative groups of agents, for several reasons:

- The centralized scheduling agent becomes a possible single point of failure that can cause the entire system to fail (unlike the decentralized GPGP system).
- The centralized scheduling agent requires a complete, global view of the episode—a view that we mentioned earlier is not always easy to achieve. We do not account for any costs in building such a global view in the reference implementation (viewing it as an upper bound on performance). We do not allow dynamic changes in the episodic task structure (which might require rescheduling).
- The centralized reference scheduler uses an *optimal* single-agent schedule as a starting point. The problem of scheduling actions in even fairly simple task structures is in NP, and the optimal scheduler's performance grows exponentially worse with the number of methods to be scheduled. Since the centralized reference scheduler has a global view and schedules all actions at all agents, the size of the centralized problem always grows faster than the size of the scheduling problems at GPGP agents with only partial views and heuristic schedulers.

We conducted 300 paired response experiments, using the three algorithms. "Balanced" refers to all mechanisms being on, with partial non-local views and communication of committed results and completed task groups. "Simple" refers to all mechanisms being off, with no non-local view and broadcast communication of all results. "Parallel" refers to the centralized reference scheduler that uses a heuristic parallelization of an optimal single agent schedule using a complete global view. The experiments were based on the same environmental parameters as the facilitation experiments (Table 1). There are several important things to note about this class of environments:

- The size of the episodes was kept artificially small so that the centralized reference scheduler could find an optimal schedule in a reasonable amount of run time.
- The experiments had very low (10%)numbers of **enables** relationships and a low (20%) number of MIN quality accrual functions because they penalize the simple algorithm—we demonstrate this in Section 4.4.
- Deadline pressure was also kept low (it also makes the simple algorithm perform badly).

In our experiments, the centralized parallel scheduler outperformed our distributed, GPGP agents 57% of the time (36% no difference, 7% distributed was better) using the total final quality as the only criterion. The GPGP agents produced 85% of the quality that the centralized parallel scheduler did, on average. These results need to be understood in the proper context—the centralized scheduler takes much more processing time than the distributed scheduler and cannot be scaled up to larger numbers of methods or task groups. The centralized scheduler also starts with a global view of the entire episode. Table 3 shows the results for all four measured criteria by summarizing within-block (paired-response) comparisons. For total final quality and number of deadlines missed, "better" simply refers to an episode where the algorithm in question had a greater total final quality or missed fewer deadlines, respectively. With respect to method execution time (a measure of system load) and termination time, "better" refers to the fact that one algorithm produced both a higher quality and missed fewer deadlines than the other algorithm, or if the two algorithms were the same, then the better algorithm had a lower total method execution time (lower load) or terminated sooner.[6]

We also looked at performance without any of the mechanisms; on the same 300 episodes the GPGP agents produced on average 1.14 times the final quality of the uncoordinated agents. Coordinated agents ("balanced") execute far fewer methods because of their ability to avoid redundancy. The redundant execution of methods proves a much more hindering element to the uncoordinated agents when acting under severe time pressure [4]. Table 4 summarizes the results.

	Parallel better	**Balanced Better**	**Same**	*Significant?*
Total Final Quality	57%	7%	36%	yes
Method Execution Time	80%	7%	13%	yes
Deadlines Missed	1%	1%	98%	no
Termination Time	67%	15%	18%	yes

Table 3: Performance comparison: Centralized Parallel Scheduler vs. Balanced GPGP Coordination and Decentralized DTT Scheduler

	Simple better	**Balanced Better**	**Same**	*Significant?*
Total Final Quality	8%	21%	71%	yes
Method Execution Time	12%	72%	16%	yes
Deadlines Missed	0%	4%	96%	yes
Termination Time	9%	58%	33%	yes

Table 4: Performance comparison: Simple GPGP Coordination vs. Balanced GPGP Coordination

[6]Termination within two time units was considered "the same" because the "balanced" algorithm has a fixed 2-unit startup cost. The average task duration is 10 time units.

4.3 Taking Advantage of a Coordination Relationship: When to Add a New Mechanism

A practical question to ask is simply whether the addition of a particular mechanism will benefit performance for the system of agents. Here we give an example with respect to the soft coordination mechanism (Mechanism 5), which will make commitments to facilitation relationships. We ran 234 randomly generated episodes (generated with the environmental parameters shown in Table 1) with four agents both with and without the soft coordination mechanism. Because the variance between these randomly generated episodes is so great, we took advantage of the paired response nature of the data to run a non-parametric Wilcoxon matched-pairs signed-ranks test [3]. This test is easy to compute and makes very few assumptions—primarily that the variables are interval-valued and comparable within each block of paired responses. For each of the 234 blocks we calculated the difference in the total final quality achieved by each group of agents and excluded the blocks where there was no difference, leaving 102 blocks. We then replace the differences with the ranks of their absolute values, and then replace the signs on the ranks. Finally we sum the positive and negative ranks separately. A standardized Z score is then calculated. A small value of Z means that there was not much consistent variation, while a large value is unlikely to occur unless one treatment consistently outperformed the other. In our experiment, the null hypothesis is that the system with the soft coordination mechanism did the same as the one without it, and our alternative is that the system with the soft coordination mechanism did better (in terms of total final quality). The result here was $Z = -6.9$, which is highly significant, and allows us to reject the null hypothesis that the mechanism did not have an effect.

4.4 Different Family Members for Different Environments

In this section we show a particular example of how different family members do better and worse in different environments. We will concentrate on two distinct family members—the 'modular agent' archetype (all CR modules on, non-local views, communicate commitments and completed task groups), and the 'simple agent' (no CR modules on, no non-local views, broadcast all completed methods). The environmental parameter we will vary (derived from the screening data collected in Section 5) is *QAF-min*, the percentage of tasks that have *min* as their quality accumulation function ('AND' semantics). Our hypothesis was that the modular agents would do better than the simple agents as QAF-min increased (as more tasks needed to be done). We ran 250 paired-response experiments at 5 levels of QAF-min (0, 0.25, 0.5, 0.75, 1.0) with enables-probability varying also at the same 5 levels, no time pressure, overlaps of 0.5, 5 task groups, and 4 agents per run. The performance (in terms of total final quality) of the two coordination styles was significantly different by the Wilcoxon matched-pairs signed-ranks test (199 different pairs, $Z = -3.27$, $p \leq 0.0005$). More interestingly, we can see the difference in performance widening with the value of QAF-min. Figure 5 shows the probability of one coordination style or the other doing better (calculated simply from the frequencies) plotted verses the value of QAF-min. This allows you to see graphically the difference in the styles as QAF-min changes.

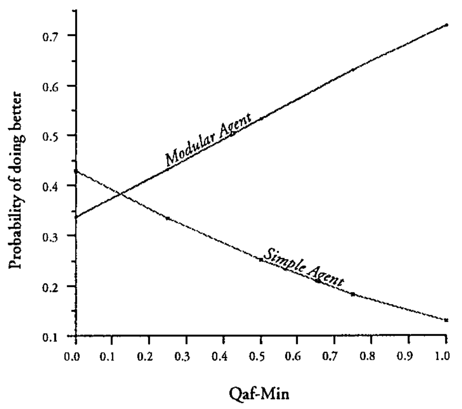

Figure 5: Plot of the probability of the modular or simple coordination styles doing better than the other (total final quality) verses the probability of task quality accumulation being MIN (AND-semantics)

4.5 Meta-level Communication: Return to Load Balancing through Dynamic Reorganization

Another question we have examined is the effect of task structure variance on the performance of load balancing algorithms. This work is a logical follow-on to the analysis of static, dynamic, and negotiated reorganization detailed in [6]. A *static* organization divides the load up *a priori*—in the case below, by randomly assigning redundant tasks to agents. A *one-shot dynamic* reorganization, like that analyzed in [7], assigns redundant tasks on the basis of the *expected* load on other agents. A *meta-level communication* (MLC) reorganization assigns redundant tasks on the basis of actual information about the particular problem-solving episode at hand. Because it requires extra communication, the MLC reorganization is more expensive, but the extra information pays off as the *variance* in static agent loads grows.

A MLC coordination mechanism (mechanism 6) can be implemented in GPGP. Many such implementations are possible; the one that we chose works by altering the way redundant commitments are handled. When a commitment is sent to another agent, it is modified to include the current *load* of the agent making the commitment (to be precise, the amount of work for the agent in the current schedule). Whenever a decision about redundant commitments need to be made at another agent (in mechanisms 3, 4, and 5—simple redundancy, hard, and soft successor relationship handling) the load of the agents with the redundant commitments are taken into account at the point where ties would have been broken randomly. The agent with the lowest load keeps the commitment instead. If the loads are equal, the tie is broken randomly as before.

The effect of this mechanism on the general GPGP environments when agents use the default Design-To-Time scheduler is minimal. The heuristics used by the DTT scheduler are focused at providing the highest possible total final quality for the agent without violating deadlines—this is not the same as terminating quickly, and the scheduler has no heuristics to prefer earlier termination times (nor, frankly, should it have them). In a randomly-generated task environment, where the methods are assigned to agents randomly (and therefore, somewhat evenly) there is rarely any significant change in termination time.

However, if you recall one of our results from [6, 7], you will remember that MLC coordination is most useful in environments with high *variance* in the task structures presented to agents. We can look at our experiments in this light, by calculating an endogenous input variable for each run that represents the amount of variance in redundant tasks (the ones that would *potentially* be eligible for a load-balancing mechanism decision). Figure 6 shows how the probability of terminating more quickly with the MLC load balancing algorithm grows as the standard deviation in the total durations of redundant tasks at each agent grows.

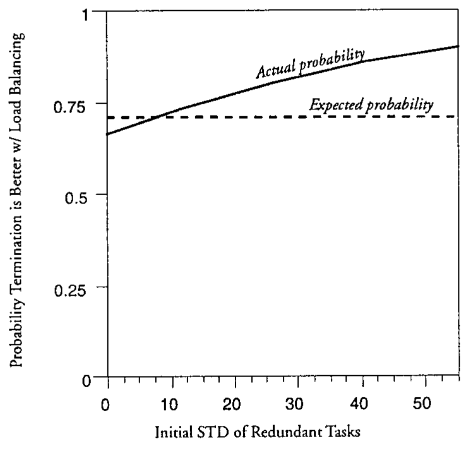

Figure 6: Probability that MLC load balancing will terminate more quickly than static load balancing, fitted using a loglinear model from actual TÆMS simulation data.

5 Exploring the Family Performance Space

Finally, we looked at the multidimensional performance space for the family of coordination algorithms over four different performance measures. At the most abstract level, each of the five mechanisms are parameterized independently (the first two have three possible settings and the last three can be 'in' or 'out') for a total of 72 possible coordination algorithms. We applied two standard statistical clustering techniques to develop a much smaller set of significantly different algorithms. The resulting five 'prototypical' combined behaviors are a useful starting point when searching for an appropriate algorithm family member in a new environment.

The analysis proceeded as follows: we generated one random episode in each of 63 randomly chosen environments, and ran each of the 72 "agent types" on the episode (4536 cases). We collected four performance measures: total quality, number of methods executed, number of communication actions, and termination time. We then took this data and standardized each performance measure within an environment. So now each measure is represented as the number of standard deviations from the mean value in that environment. We then took summary statistics for each measure grouped by agent types—this boils the 4536 cases (standardized within each environment) into 72 summary cases (summarized across environments). Each of the 72 summaries correspond to the average standardized performance of one agent-type for the four performance measures. We then used both a hierarchical clustering algorithm

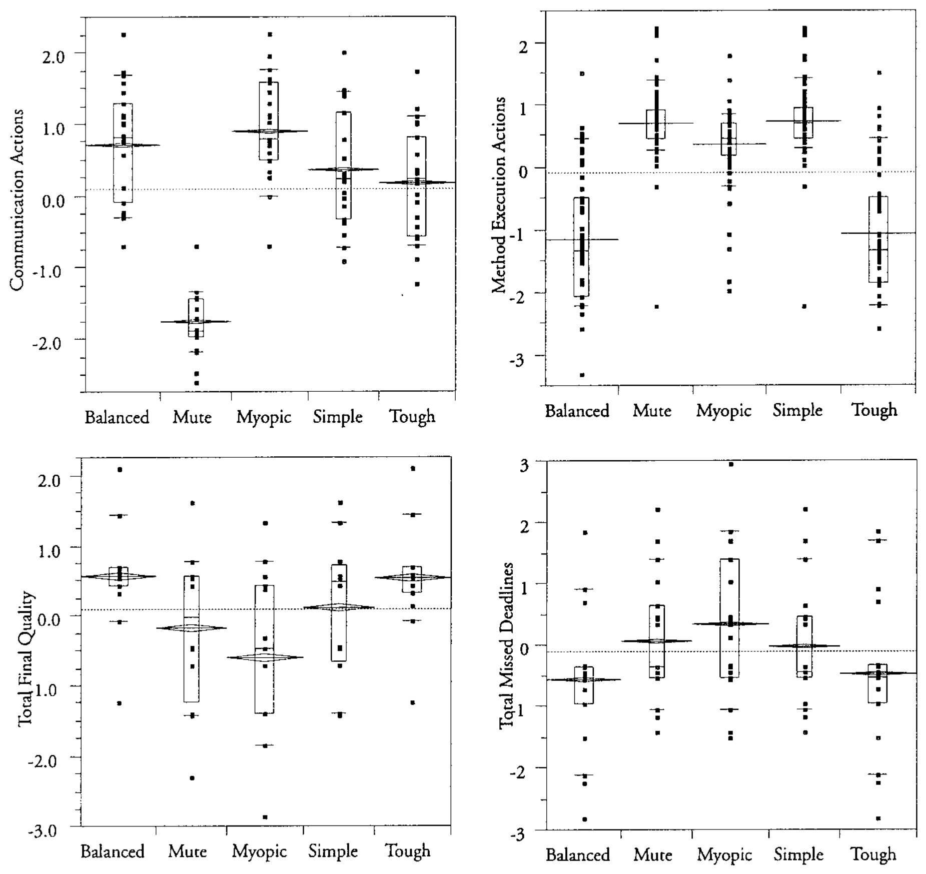

Figure 7: Standardized Performance by the 5 named coordination styles.

(SYSTAT JOIN with 'complete linkage'[7]) to produce the following general prototypical agent classes (we chose one representative algorithm in each class):

Simple: No commitments or non-local view, just broadcasts results.

Myopic: All commitment mechanisms on, but no non-local view.

Balanced: All mechanisms on.

Tough-guy: Agent that makes no soft commitments.

Mute: No communication whatsoever[8]

Figure 7 shows the values of several typical performance measures for only the five named types. Performance measures were standardized *within each episode*, (i.e. across all 72 types). Shown for each are the means and 10, 25, 50, 75, and 90 percent quantiles. All algorithms' performances are significantly different by Tukey Kramer HSD except for: Method Execution

[7]Distances are calculated between the farthest points in each cluster. Other distance measures (Euclidean, centroid, or Pearson correlation) gave similar results.

[8]This algorithm makes no commitments (mechanisms 3, 4, and 5 off) and communicates (mechanism 2) only 'satisfied commitments'—therefore it sends no communications ever!.

(Simple vs. Mute), Total final quality (Balanced vs. Tough), Deadlines missed (simple vs. mute) and (balanced vs. tough).

We are also analyzing the effect of environmental characteristics on agent performance. Figure 8 shows an example of the effect of the amount of overlap (method redundancy) on the number of method execution actions for the five named agent types. Note again that the balanced and tough agents do significantly less work when there is a lot of overlap (as would be expected). The performance of the tough and balanced agents is similar because (from Table 1) 1) the algorithms only differ in the way that they handle facilitation, and 2) only half the experiments had any facilitation, and when it was present was only at 50% power.

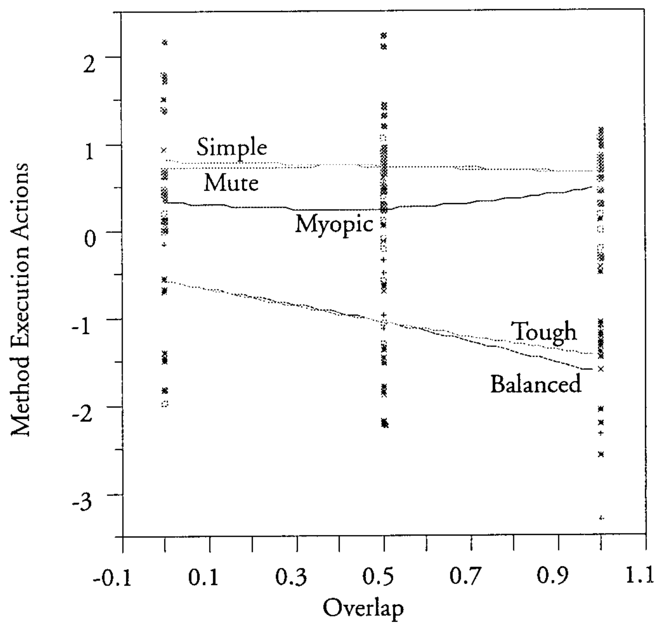

Figure 8: The effect of overlaps in the task environment on the standardized method execution performance by the 5 named coordination styles (smoothed splines fit to the means).

A linear clustering algorithm, SYSTAT KMEANS, produces a similar result as hierarchical clustering, and also produces the mean value of each performance measure for each group. For example, the non-communicating agents have a high negative mean "number-of-communications" (-1.16; remember these were averaged from standardized scores) but execute more methods on average and produce less final quality. They also terminate slightly quicker than average. The "balanced" group, in comparison, communicates a little more than average, executes *many* fewer methods (-1.29—way out on the edge of this statistic), returns better-than-average quality and about average termination time. This is reasonable, as 'avoiding redundant work' and other work-reducing ideas are a key feature of the original PGP algorithm replicated by this set of mechanisms.

6 Conclusions and Future Work

This paper discusses the design of an extendable family of scheduling coordination mechanisms, called Generalized Partial Global Planning (GPGP), that form a basic set of coordination mechanisms for teams of cooperative computational agents. An important feature of this approach includes an extendable set of modular coordination mechanisms, any subset or all of which can be used in response to a particular task environment. This subset may be parameterized, and the parameterization does not have to be chosen statically, but can instead be chosen on a task-group-by-task-group basis or even in response to a particular problem-solving situation. For example, Mechanism 5 (Handle Soft Predecessor CRs) might be "on" for certain classes of tasks and 'off' for other classes (that usually have few or very weak soft CRs). The general specification of the GPGP mechanisms involves the detection and response to certain abstract *coordination relationships* in the incoming task structure that were not tied to a particular domain. We have used TÆMS to model a simple distributed sensor network problem, the original DVMT domain, and a hospital scheduling environment. A careful separation of the coordination mechanisms from an agent's local scheduler allows each to better do the job for which it was designed. We believe this separation is not only useful for applying our coordination mechanisms to problems with existing, customized local schedulers, but also to problems involving humans (where the coordination mechanism can act as an interface to the person, suggesting possible commitments for the person's consideration and reporting non-local commitments made by others).

The GPGP coordination approach as described in this paper has been fully implemented in the TÆMS simulation testbed. Significant experimental validation of the GPGP approach is documented in [4]. This paper showed how to decide if the addition of a new GPGP mechanism was useful. It showed the general performance of two GPGP family algorithms compared to a centralized parallel reference algorithm; GPGP with all mechanisms on produces 85% of the quality of the centralized reference scheduler in a random environment. Such performance is reasonable and we feel could be made even better by developing better local scheduling algorithms and new coordination mechanisms.

We also demonstrated how a feature of the task environment (the probability of task quality accumulation being MAX) can cause different GPGP family members to be preferred. We also discussed a sixth mechanism, a load balancing mechanism that communicates meta-level information, and showed that it was somewhat more useful when the variance in duration of the agents' overlapping tasks was high. This section thus ties-in back to the discussion in [6, 7] on the usefulness of meta-level communication (in this case, the transmission of local load information) when the inter-episode variance (in this case, in the initial agent loads) is high.

Finally, we gave a sense of the performance space of the five broadly-parameterized mechanisms using a clustering technique. Clustering can be a useful method for dealing with large algorithm spaces to prune search for an appropriate combination of mechanisms. Such methods may also lead to ways to learn situation-specific knowledge about the application of certain mechanisms in certain situations (perhaps using case-based reasoning techniques).

We believe that GPGP can become a reusable, domain-independent basis for multi-agent coordination when used in conjunction with a library of coordination mechanisms and a learning mechanism. We intend to develop such a library of reusable coordination mechanisms. For example, mechanisms that work from the successors of hard and soft relationships instead of the predecessors, negotiation mechanisms, mechanisms for behavior such as contracting, or mechanisms that can be used by self-motivated agents in non-cooperative environments. Many of these mechanisms can be built on the existing work of other DAI researchers. Future work will also examine expanding the parameterization of the mechanisms and using machine learning techniques to choose the appropriate parameter values (i.e., learning the best mechanism set for an environment). Finally, we are also beginning work on using the GPGP approach in applications ranging from providing human coordination assistance to distributed information gathering.

References

[1] C. Castelfranchi. Commitments:from individual intentions to groups and organizations. In Michael Prietula, editor, *AI and theories of groups & organizations: Conceptual and Empirical Research*. AAAI Workshop, 1993. Working Notes.

[2] Philip R. Cohen and Hector J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(3):213–261, 1990.

[3] W. W. Daniel. *Applied Nonparametric Statistics*. Houghton-Mifflin, Boston, 1978.

[4] Keith S. Decker. *Environment Centered Analysis and Design of Coordination Mechanisms*. PhD thesis, University of Massachusetts, 1995.

[5] Keith S. Decker and Victor R. Lesser. Generalizing the partial global planning algorithm. *International Journal of Intelligent and Cooperative Information Systems*, 1(2):319–346, June 1992.

[6] Keith S. Decker and Victor R. Lesser. An approach to analyzing the need for meta-level communication. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 360–366, Chambéry, France, August 1993.

[7] Keith S. Decker and Victor R. Lesser. A one-shot dynamic coordination algorithm for distributed sensor networks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 210–216, Washington, July 1993.

[8] Keith S. Decker and Victor R. Lesser. Quantitative modeling of complex computational task environments. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 217–224, Washington, July 1993.

[9] E. H. Durfee and T. A. Montgomery. Coordination as distributed search in a hierarchical behavior space. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1363–1378, November 1991.

[10] E.H. Durfee and V.R. Lesser. Partial global planning: A coordination framework for distributed hypothesis formation. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(5):1167–1183, September 1991.

[11] E. Ephrati and J.S. Rosenschein. Divide and conquer in multi-agent planning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 375–380, Seattle, 1994. AAAI Press/MIT Press.

[12] Alan Garvey, Keith Decker, and Victor Lesser. A negotiation-based interface between a real-time scheduler and a decision-maker. CS Technical Report 94–08, University of Massachusetts, 1994.

[13] Alan Garvey and Victor Lesser. Design-to-time real-time scheduling. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6):1491–1502, 1993.

[14] B. Grosz and S. Kraus. Collaborative plans for group activities. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Chambéry, France, August 1993.

[15] N. R. Jennings. Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review*, 8(3):223–250, 1993.

[16] V. R. Lesser. A retrospective view of FA/C distributed problem solving. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1347–1363, November 1991.

[17] Thomas W. Malone and Kevin Crowston. Toward an interdisciplinary theory of coordination. Center for Coordination Science Technical Report 120, MIT Sloan School of Management, 1991.

[18] Y. Shoham and M. Tennenholtz. On the synthesis of useful social laws for artificial agent societies (preliminary report). In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 276–281, San Jose, July 1992.

[19] Yoav Shoham. AGENT0: A simple agent language and its interpreter. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 704–709, Anaheim, July 1991.

[20] Frank v. Martial. *Coordinating Plans of Autonomous Agents.* Springer-Verlag, Berlin, 1992. Lecture Notes in Artificial Intelligence no. 610.

APPENDIX B

Understanding the Role of Negotiation in Distributed Search Among Heterogeneous Agents

Susan E. Lander and **Victor R. Lesser**
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
{lander,lesser}@cs.umass.edu

Abstract

In our research, we explore the role of negotiation for conflict resolution in distributed search among heterogeneous and reusable agents. We present *negotiated search*, an algorithm that explicitly recognizes and exploits conflict to direct search activity across a set of agents. In negotiated search, loosely coupled agents interleave the tasks of 1) local search for a solution to some subproblem; 2) integration of local subproblem solutions into a shared solution; 3) information exchange to define and refine the shared search space of the agents; and 4) assessment and reassessment of emerging solutions.

Negotiated search is applicable to diverse application areas and problem-solving environments. It requires only basic search operators and allows maximum flexibility in the distribution of those operators. These qualities make the algorithm particularly appropriate for the integration of heterogeneous agents into application systems. The algorithm is implemented in a multi-agent framework, TEAM, that provides the infrastructure required for communication and cooperation.

1 Introduction

The current state of knowledge-based technology is such that almost every application system is built from scratch. In order to move beyond the prohibitive cost of constantly reinventing, rerepresenting, and reimplementing the wheel, researchers are beginning to examine the feasibility of building application systems with reusable agents [Neches *et al.*, 1991]. A reusable agent is designed to work without *a priori* knowledge of the agent set in which it will be embedded, instead using a flexible, reactive approach to cooperation. Although this flexibility can lead to inefficient problem solving, an agent can often gather information about the agent set as problem solving progresses to improve efficiency.

This research was supported by ARPA under ONR Contract #N00014-92-J-1698. The content of the information does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred.

Multi-agent systems do not traditionally acknowledge the role of conflict among agents as a driving force in the control of problem-solving activity. In reusable-agent systems, however, conflict is inevitable since agents are implemented at different times by different people and in different environments. We present a distributed-search algorithm, *negotiated search*, that uses conflict as a source of control information for directing search activity across a set of heterogeneous agents in their quest for a mutually acceptable solution.

The negotiated-search algorithm has been successfully incorporated into two implemented systems. In [Lander and Lesser, 1992b], we describe distributed search in the context of a seven-agent steam condenser design system and discuss how different operator/agent assignments within the negotiated-search algorithm affect problem solving. In [Lander and Lesser, 1992a], a two-agent contract negotiation system is presented, and negotiated search is compared to a search strategy that is tailored to characteristics of that environment. Through analysis of the environment and search algorithms, we show the versatility and effectiveness of negotiated search in reusable-agent systems while also pointing out that customized search strategies are inflexible but can improve system performance when they can be applied. In this paper, we describe negotiated search from an application-independent perspective.

The need for a flexible algorithm to support reusability and heterogeneity motivates particular aspects of negotiated search:

- Conflict, negotiation, and democratic determination of acceptability are integral parts of the algorithm.
- Agent coordination is accomplished through clearly defined individual roles in the evolution of a shared solution. These roles are realized as operators that accomplish state transitions on shared solutions.
- Operators represent standard and widely available search and information-assimilation capabilities. A particular agent may instantiate all defined operators or some subset of defined operators.
- Whenever possible, feedback is used to refine the perceived search spaces of individual agents to more closely reflect the true composite search space.

TEAM agents are not hostile and will not intentionally mislead or otherwise try to sabotage another agent's reasoning. They are cooperative in the sense that an agent is willing to contribute both knowledge and solutions to other agents as appropriate and to accept solutions that are not locally optimal in order to find a mutually-acceptable solution. Each agent is a stand-alone system with specific capabilities that allow it to be included in an integrated multi-agent system. We assume that agents can be heterogeneous in architecture, inference engines, evaluation criteria and priorities for solutions, and in long-term knowledge. Each agent does its own internal scheduling and has private data, knowledge, and history mechanisms.

In negotiated search, agents interleave the tasks of 1) local search for a solution to some subproblem; 2) integration of local subproblem solutions into a shared solution (the *composite* solution);[1] 3) negotiation to define and refine the shared search space of the agents; and 4) assessment and reassessment of emerging solutions.

In the remainder of the paper, we first motivate the development of our negotiated-search model by presenting an intuitive description of negotiation and, from this foundation, constructing an algorithmic model of the negotiation process. The next section details negotiated search from a state-based perspective similar to that used by von Martial to describe negotiation protocols in distributed planning [von Martial, 1992]. We then present seven basic negotiated-search operators. The final section briefly describes the status of the implementation and extensions to this model that are not covered in this paper.

2 An Initial Perspective on Negotiation

In this section, we begin with an intuitive description of negotiation:

> One agent generates a proposal and other agents review it. If some other agent doesn't like the proposal, it rejects it and provides some feedback about what it doesn't like. Some agent may generate a counter-proposal. If so, the other agents (including the agent that generated the first proposal) then review the counter-proposal and the process repeats. As information is exchanged, conflicts become apparent among the agents. Agents may respond to the conflicts by incrementally relaxing individual preferences until some mutually acceptable ground is reached.

This example captures the primary characteristics that one would expect to see:

- proposals are generated by one or more agents
- agents evaluate proposals based on their individual criteria for solution acceptability
- agents provide feedback about what they like or don't like about particular proposals, resulting in a progressively better understanding of the shared requirements for solutions over time
- agents can play different roles in the negotiation process, e.g., an agent can be a reviewer for another agent's proposal and then be a generator for a counter-proposal
- conflicts exist among the agents' requirements for acceptable solutions
- agents incrementally relax their solution requirements to reach agreement
- the decision to accept or not accept a proposal is a joint, democratic process

Some extensions to the definition are required. For example, it assumes that a proposal becomes a solution when it is accepted by all agents. However, this assumption rules out situations in which high-level problems are decomposed and each agent works on some subproblem. In this case, the proposal an agent makes does not represent a complete solution but rather some component of a solution that interacts with other components through shared attributes. Evaluation is then indirect since an agent cannot evaluate proposals for interacting components that are outside of its domain of expertise. In negotiated search, an agent evaluates an external interacting-component proposal by creating and evaluating a compatible local proposal (i.e., one that has the same values for shared attributes), thereby focusing on how the external proposal affects local quality.

Although a proposal includes the information required to implement a solution, it provides only a surface-level view of the reasoning that went into creating it. It is sometimes possible to make guesses about other agents' requirements that could be used in generating counter-proposals. However, in the general case of reusable agents, external local evaluation criteria for solutions cannot be predicted, nor can they be inferred from the "snapshot" provided by a proposal. For proposals and counter-proposals to be related, there must be a deeper understanding of the shared search space of the agents. This understanding is achieved through a feedback system that can be separate from the proposals.

3 Negotiated Search

Artificial intelligence researchers have previously used the term negotiation with respect to conflict resolution and avoidance [Adler *et al.*, 1989, Klein, 1991, Lander and Lesser, 1992a, Sycara, 1985, Werkman, 1992], task allocation [Cammarata *et al.*, 1983, Durfee and Montgomery, 1990, Davis and Smith, 1983], and resource allocation [Adler *et al.*, 1989, Conry *et al.*, 1992, Sathi and Fox, 1989, Sycara *et al.*, 1991]. Negotiation is sometimes treated as an independent process that is used to select one of a set of existing alternative solutions [Zlotkin and Rosenschein, 1990] rather than as an inherent part of a solution-generation process. It can be difficult under conditions where agents are hostile and unwilling to share private information [Sycara, 1985]. Negotiation can occur among peers [Cammarata *et al.*, 1983, Lander and Lesser, 1992b], through a mediator or arbitrator [Sycara, 1985, Werkman, 1992], or hierarchically through an organization [Durfee and Montgomery, 1990, Davis and Smith, 1983]. It can occur at

[1]Sathi similarly uses the term *composition* as the name of a specific search operator that combines local information [Sathi and Fox, 1989]

either the domain or control level of problem-solving. Laasri et. al. describe the *recursive negotiation model*, a general model of multi-agent problem solving that details various situations that can potentially benefit from negotiation [Laasri *et al.*, 1992]. In examining this model, it becomes clear that negotiation is a pervasive process that remains relatively untapped by current computational systems. In developing the negotiated-search model, we have tried to capture the key requirements for negotiation without restricting the domain, task decomposition, or organizational model of the agent set.

Several researchers have developed algorithms and heuristics for constraint-directed distributed search in situations involving multiple homogeneous agents [Sathi and Fox, 1989, Sycara *et al.*, 1991, Yokoo *et al.*, 1992].[2] We extend this work to handle situations where heterogeneous agents may have different or multiple local problem-solving paradigms, instantiate different search operators, and where agents may not be able to provide specific information to other agents or understand information received from other agents. The negotiated-search algorithm is particularly suitable to this style of problem solving because 1) the required search operators represent standard search capabilities; 2) the search operators can be flexibly assigned across the agent set according to the search capabilities of each agent; and 3) agents use incremental relaxation of solution requirements to reach mutual acceptability as an inherent part of problem solving.

3.1 The Search Process

Search is initiated by a problem specification that details the form of a solution and values, preferences, or constraints on some attributes of that solution. This specification is placed in a centralized shared memory as are emerging composite solutions.[3] Some agent(s) uses constraining information from the specification and its local solution requirements to propose an initial partial solution called a *base proposal*. The base proposal is then extended and evaluated by other agents during future processing cycles. When a particular solution cannot be extended by some agent due to conflicts with existing solution attributes, there are two possible outcomes: 1) if the conflict is caused by the violation of some hard (non-relaxable) requirement, the solution path is pruned (e.g., arc 5 in Figure 1); or 2) if the conflict is caused by the violation of some soft (relaxable) solution requirement, the solution is saved and viewed as a potential compromise (e.g, arc 9 in Figure 1). In the first case, no more work will be done on that solution, and, to the extent that the violated requirement can be communicated to and assimilated by other agents, future counter-proposals will not violate that same requirement. In the second case, the violated requirement may eventually be relaxed and, if that happens, the potential compromise will become a viable solution again. Future counter-proposals will take the violated requirement into account but are not guaranteed to avoid the same conflict, since other alternatives may be worse.

In both of the above cases, conflict is used as the trigger for the communication of feedback information. In multi-agent systems, it is always problematic to decide what information should be exchanged and when that exchange should take place. In general, agents want to minimize the amount of information they share since it is expensive both to communicate information and to assimilate information. On the other hand, sharing information that will specifically help another agent avoid future conflicts is generally cost effective since it eliminates the expense of generating unproductive solution paths [Lander, 1993]. In negotiated search, an agent that receives conflict information from another agent can choose whether or not to prune its own search to respect that information (see Section 4.5).

Multiple solution paths can be concurrently investigated in negotiated search. Agents are free to initiate solutions at any time either because there aren't any promising solutions in the current solution set or because they have no other work to do. Advantages to maintaining multiple paths include exploiting the potential for concurrent activity and having the ability to directly compare different potential compromises. There are disadvantages to concurrently exploring multiple solution paths however: there will be multiple partial solutions that have to be stored at all times, requiring additional memory resources. There is also overhead involved in focusing on a promising solution path at a particular point in problem solving, both from the local and global perspectives, and in managing the links between solution components along each path. The number of open solution paths is highly dependent on the domain, the number of agents, and the control policies of individual agents. This number can be controlled through parameter settings in TEAM and through the specification of which negotiated-search operators will be active for each agent in the agent set.

3.2 A State-Based View of Negotiated Search

Figure 1 provides a state-based view of the transition of a composite (shared) solution from its initial state (a problem specification) to a termination state (an infeasible solution, an unacceptable solution, or a complete acceptable solution). In this figure, states are defined in terms of three attributes of composite solutions: *acceptability*, *completeness*, and *search-state*. The possible values for *acceptability* are *acceptable*, *unacceptable*, and *infeasible*. Possible values for *completeness* are *complete* and *incomplete*. Note that *complete* means that all agents have had the opportunity to extend or critique the solution. A solution with all required components can still be waiting for critiques from other agents and is not considered complete in that case. *Search-state* can take the values *initial* or *closed*.

A *negotiated-search operator* is a search function applied by an agent. Each operator has a generic form that is expressed in an agent language defined by TEAM, spec-

[2] Agents may control different resources and have different constraints on solutions, but they share a single underlying problem-solving paradigm and knowledge representation.

[3] Each agent also has a local short-term memory where it stores intermediate results and/or component proposals that are linked to composite solutions in shared memory.

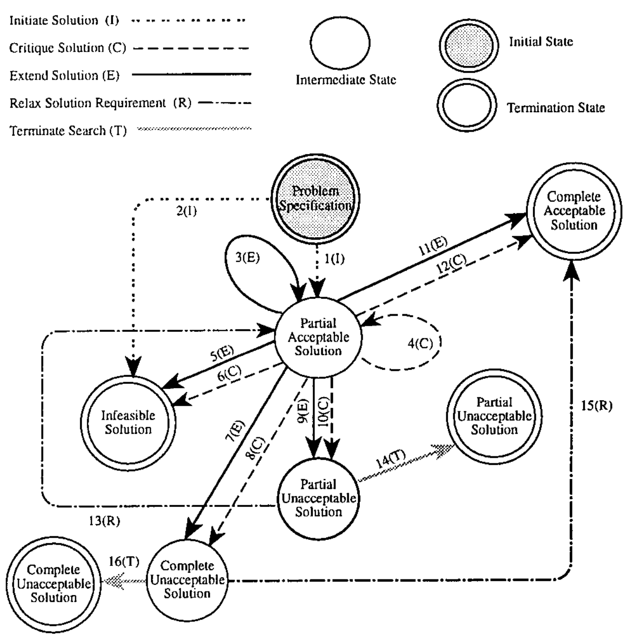

Figure 1: *A State-Based View of Negotiated Search*

ifying its inputs, outputs, and functionality. The decision to apply a particular operator to a problem-solving situation is made by an agent within its local view of the problem-solving situation. The arcs in Figure 1 are negotiated-search operators that can be applied by some agent to a solution.

Each agent instantiates one or more of the negotiated-search operators: *initiate-solution*, *extend-solution*, *critique-solution*, and *relax-solution-requirement.* In addition, TEAM instantiates the *terminate-search* operator. These operators will be described in detail below, but we provide an overview here to provide a sense of their functionality. *Initiate-solution* is applied by an agent to generate a *base proposal* that will be used as the basis for a new composite solution. *Extend-solution* is applied by an agent to: 1) add a component proposal to a composite solution; 2) evaluate the composite solution from a local perspective; and 3) provide feedback information if conflicts are detected. *Critique-solution* is applied to: 1) evaluate a composite solution (without generating a component proposal); and 2) provide feedback information if conflicts are detected. *Relax-solution-requirement* is applied to: 1) select a local requirement to relax; 2) update the local database to effect the relaxation; and 3) reevaluate existing solutions in light of the relaxation. *Terminate-search* is applied by TEAM to change the state of the problem solving from *initial* to *closed*, thereby changing the termination status of solutions.

The negotiated-search algorithm is applied by a set of agents, $\mathcal{A}$. Let $\mathcal{A} = \{A1, A2, A3\}$ and assume that $A1$ initiates a solution, $A2$ extends the solution, and $A3$ critiques some aspect of that solution. We examine a typical search in which a conflict occurs. $A1$ first applies the operator *initiate-solution* to a problem specification and produces a partial acceptable solution (*arc 1*). Then $A2$ applies *extend-solution* without detecting a conflict. Although the solution now has all components specified, it is not *complete* until all critiques have also been received. Therefore the solution is now partial and acceptable (*arc 3*) . $A3$ next applies *critique-solution*, detects a conflict, and evaluates the solution as unacceptable (*arc 8*). This solution remains as it is for some amount of time while the agents are working on other solution paths. When further search fails to produce an acceptable solution, $A3$ decides to relax the requirement that made this solution unacceptable. The solution is now acceptable to $A3$ and, since it was already complete, it reaches the termination state of complete acceptable solution (*arc15*). In this way, various paths through the state diagram can be achieved by the agent set.

Although the above example describes a sequential ordering of operator applications, TEAM permits concurrency except where there are domain-dependent operator preconditions that force sequential execution. Concurrency requires that TEAM have mechanisms for handling conflicts that occur due to the simultaneous development of extending proposals and criticisms. These mechanisms are discussed in [Lander, 1993].

4 Negotiated Search Operators

In this section, we present a detailed description of the negotiated-search operators. Notice that the operators depicted in Figure 1 work at the surface level of problem solving: they move a particular solution through various states to a termination state. They do not address the issue of feedback and its effect on problem solving. Later in this section, we will present two operators that an agent applies to assimilate conflict information into its knowledge base, thereby refining its view of the search space.

4.1 Initiate-Solution

Initiate-solution is the basic operator for initiating solutions. It is applied within the agent's view of solution requirements: local requirements, those imposed by the problem specification, and any known external requirements learned from other agents. Given these requirements, it creates the *base proposal. Initiate-solution* is executed by one or more agents at system start-up time, and may be repeatedly executed as earlier proposed solutions are rejected by other agents or if alternative solutions are desired. If earlier solutions have been proposed and rejected, the initiating agent may have received conflict information that will influence the generation of new base proposals.

At least one agent must instantiate *initiate-solution*; however, instantiating it at multiple agents is likely to result in a more diverse set of solution paths and more thorough coverage of the composite solution space. Depending on characteristics of the agents and agent set, it may also have a distracting effect. Trade-offs between coverage and distraction are a ubiquitous problem in distributed systems and are discussed generally in [Lesser and Erman, 1980] and specifically with respect to negotiated search in [Lander and Lesser, 1992b].

When no base proposal can be found under the existing set of requirements, an agent can relax requirements to expand the search space. If there are requirements on solutions that come from information communicated by another agent (external requirements), the initiating agent can ignore one or more of these requirements in its own search. Notice that the other agent does not actually relax the requirements. In this way, each agent chooses the set of requirements, both internal and external, it will attempt to satisfy. When known external requirements are violated, the proposal is suggested as a possible compromise rather than a fully acceptable solution. The external agent that has its requirements violated in the compromise proposal cannot be forced to accept it. Because the selection of a mutually acceptable solution is democratic, each agent votes on the acceptability of a solution. The external agent that has the violated requirement(s) can initially vote that the solution is unacceptable but, if it does not find a better alternative, it may eventually agree to accept this compromise.

If there are no relaxable external solution requirements or if the external requirements are inflexible, an agent can relax some local requirement. If no base proposal can be found at any level of external or internal requirement relaxation, the agent returns a failure along with any conflict information it can generate that describes why it failed. TEAM returns a failure if no agent can generate a new base proposal and all previously created solutions have been found to be *infeasible.*

4.2 Critique-Solution and Extend-Solution

The *critique-solution* operator is applied by an agent to evaluate a partially or fully specified composite solution. The *extend-solution* operator is applied by an agent to extend and evaluate a partially specified composite solution. These two operators will be described jointly because of their similarity. The input for these operators is a composite solution that was initiated by another agent. The output for *critique-solution* is an evaluation, and when a conflict is detected, conflict information. The output for *extend-solution* is a proposal, an evaluation, and, when a conflict exists, conflict information.

The *extend-solution* operator is required in domains where solutions comprise interacting components and each component is developed by an expert agent. The component that an agent develops with *extend-solution* must be compatible with the solution being extended (it must have the same values for solution variables that overlap). The agent executing the operator searches for a compatible proposal under its known solution requirements and the requirements imposed by the assigned parameter values of the solution to be extended.

Although we will not discuss *critique-solution* further, the following discussion of *extend-solution* generally applies to both operators, except that *critique-solution* evaluates the existing composite solution rather than creating and evaluating a compatible proposal. In *extend-solution*, if a compatible proposal is found that does not violate any local solution requirements, it is returned as an *acceptable* proposal. If the best compatible proposal found violates some relaxable (soft) local solution requirements (where the best proposal is one that maximizes local evaluation), it is returned as *unacceptable* along with information that describes the conflict. Although currently unacceptable, future requirement relaxations may change its status and, therefore, the solution is saved as a potential compromise. In the final case, no compatible proposal can be found without violating nonrelaxable (hard) requirements of the executing agent. In this case, the agent fails and the solution path is marked as *infeasible.* Conflict information is returned whenever possible that describes why the path is infeasible, i.e., what hard requirements were violated.

4.3 Relax-Solution-Requirement

Relaxation of solution requirements is a necessary part of negotiated search. In order to terminate problem solving, agents must reach mutual acceptability on one or more solutions. Acceptability is defined as an attribute of a composite solution as shown in Figure 1. If any agent locally evaluates a solution as unacceptable, the solution is considered globally unacceptable. However, as can be seen in that figure, a solution that is unacceptable at some point in time can later become acceptable when the agent or agents that reject it relax their solution requirements.

There are three primary forms of relaxation, *unilateral relaxation*, *feedback-based relaxation*, and *problem-state relaxation.* Unilateral relaxation occurs when an agent decides to relax a requirement due to its inability to find a solution under the problem specification, i.e., the agent finds that, given the problem specification and its initial solution requirements, it cannot produce a locally acceptable proposal. This situation occurs in the application of the *initiate-solution* operator as described in Section 4.1.

Feedback-based relaxation occurs when an agent relaxes a solution requirement because of some explicit information about the requirements of some other agent(s), i.e, a conflict is found between relaxable local solution requirements and less flexible external solution requirements. This occurs when external information has been received by an agent and is being assimilated as described in Section 4.5.

Problem-state relaxation is a reaction to the lack of overall problem-solving progress. In the current TEAM framework, problem-state relaxation occurs at specific processing-cycle intervals: for example, all agents may relax a solution requirement after 10 processing cycles. Alternatively, the user can specify the relaxation parameter separately for each agent, so that one agent may relax after 10 processing cycles while another will relax after 20 processing cycles. Problem-state relaxation occurs because the problem may be overconstrained by the full agent set. The ability to formulate, communicate, and assimilate constraining information is not guaranteed to be complete and precise across the agent set and the reality is that agents can't always determine whether the composite search space is overconstrained. Therefore, they must have some heuristic method (as well as the deterministic methods above) for deciding when it is appropriate to relax requirements.[4] Because of problem-state relaxation, we can guarantee that if any initial proposal is generated that can result in a feasible solution, either that solution will eventually become acceptable to all agents, or some other solution will become acceptable to all agents and deadlock will not occur.

4.4 Terminate-Search

The operator *terminate-search* is applied by TEAM, rather than by an agent, to change the search phase of the algorithm from *initial* to *closed* when some (user-specified) number of acceptable proposals been found.[5] As seen in Figure 1, when this change occurs, partial and complete *unacceptable* solutions move from intermediate to termination states. Any partial *acceptable* solutions are completed however to ensure that good partial solutions are not abandoned.

4.5 Assimilating Information

There are two operators associated with assimilating information at an agent: *store-received-information* and *retrieve-information*. *Store-received-information* takes conflict information from other agents, syntactically checks to see if the information already exists in the local knowledge base and, if not, stores it so that it can be retrieved. A received requirement may be indexed by various attributes including the name of the sending agent, the flexibility of the requirement, the names and acceptable values of constrained solution attributes, and, in the case of ordered solution attributes, whether the requirement defines a minimum or maximum boundary on potential values, e.g., $x > 5$.

Retrieve-information is an operator that extends or replaces an agent's default capability to retrieve relevant constraining information from its knowledge base. Because an agent's internal knowledge is expected to be locally consistent, the default retrieval mechanism generally does not handle cases where conflicts may exist in the retrieved requirements. Requirement retrieval occurs during solution initiation, extension, and criticism. The goal of the retrieval process is to find the most restrictive, but non-conflicting, set of solution requirements that constrain a solution for the current local search problem. Different types of requirements require different treatment, but to provide a concrete example of retrieval, we present the algorithm used for selecting boundary constraints on numerical solution attributes in our application systems. Potentially relevant constraints are retrieved and sorted into maximum and minimum boundary groups. The most restrictive maximum constraint (MAX) and the most restrictive minimum constraint (MIN) from each group are selected (where most restrictive means the highest value from the MIN group and the lowest value from the MAX group). Then the algorithm loops through the following sequence until a set of minimum and maximum values is found or until it is determined that no non-conflicting set exists.

LOOP: If the value of MAX is greater than or equal to the value of MIN, return MAX and MIN since a non-conflicting set has been found. Otherwise, if the flexibility of MAX is greater than the flexibility of MIN select the next most restrictive maximum constraint (MAX) and go to LOOP. Otherwise, if the flexibility of MAX is less than the flexibility of MIN, select the next most restrictive minimum constraint (MIN) and go to LOOP. Otherwise, the flexibility of MAX is equal to the flexibility of MIN. Then: if MAX is locally owned, select the next most restrictive minimum constraint (MIN) and go to LOOP. If MAX is not locally owned and MIN is locally owned, select the next most restrictive maximum constraint (MAX) and go to LOOP. If neither MAX nor MIN is locally owned, select the next most restrictive minimum constraint (MIN) and go to LOOP.

In reusable agent sets, operator diversity is expected—not every agent will instantiate every operator including the *store-received-information* and *retrieve-information* operators. Because of this, when an agent formulates and sends conflict information to another agent, there is no guarantee that the receiving agent will use that information appropriately. Therefore, although conflict information is shared willingly and cooperatively in negotiated search, agents do not depend on other agents to react in a fixed way to that information.

4.6 Agent-Level Control of Operator Application

Figure 1 describes domain-independent state preconditions that must be satisfied before an agent can apply one of its operators to a particular solution. However, because there are multiple solution paths, and because some operators are not directly involved in solution generation (e.g., *store-received-information*), an agent may have multiple operators ready to execute at any given time. The order in which an agent schedules local operators is not mandated by either TEAM or by the negotiated-search algorithm. However, because an agent's perception of the world changes over time, the order in which particular operators are executed does affect system performance and the effect of local scheduling on the overall behavior of the system should be considered. Some general policies for local scheduling are useful in most situations, i.e., agents should assimilate any new information received before initiating or critiquing solutions. The degree of sophistication required in local scheduling though is highly dependent on the application and the complexity of required interactions.

5 Conclusions

Negotiated search is a flexible and widely applicable distributed-search algorithm. It specifically addresses issues that arise in multi-agent systems comprised of reusable and heterogeneous agents. The algorithm acknowledges the inevitability of conflict among the agents, and exploits that conflict to drive agent interaction and guide local search.

[4]Using the number of processing cycles as a heuristic is a simplistic approach. More sophisticated mechanisms for applying problem-state relaxation based on characteristics of problem-solving situation, rather than on time, are discussed in [Lander, 1993].

[5]This is a simplified version of the TEAM termination policy that integrates agent acceptability and, optionally, a domain-dependent global evaluation of solutions.

Negotiated search has been implemented in TEAM, a generic framework for the integration of reusable agents, and consequently, in two application systems built on top of TEAM: STEAM (a seven-agent system for the mechanical design of steam condensers); and AGREE (a two-agent system for buy/sell contract negotiation). Testing and analysis of the algorithm within the context of the application systems is described in other work [Lander, 1993, Lander and Lesser, 1992a, Lander and Lesser, 1992b]. Results from experiments conducted with negotiated search show that the algorithm can produce high-quality solutions. They also support the claim that the algorithm is flexible enough to work in reusable-agent systems where the search operators are randomly distributed across the agent set. We see negotiated search as a default algorithm—one that will provide reasonable solutions in a reasonable amount of time without problem-specific customization. As a complementary approach to developing this general algorithm, we are developing customized algorithms that require specific agent characteristics or inter-agent relationships to exist. By taking advantage of these characteristics, it is often possible to improve solution quality and/or processing-time performance. TEAM supports the dynamic selection of a search algorithm, thereby enabling an agent set to switch to a customized algorithm if the requirements for application of the algorithm are met. This work is described in [Lander, 1993].

Acknowledgements

We thank Margaret Connell and Kevin Gallagher for their support in this project. GBB, a system integration tool from Blackboard Technology Group, was used as the basic platform on which our framework was built.

References

[Adler *et al.*, 1989] Mark R. Adler, Alvah B. Davis, Robert Weihmayer, and Ralph W. Worrest. Conflict-resolution strategies for non-hierarchical distributed agents. In Michael N. Huhns, editor, *Distributed Artificial Intelligence, Volume 2*, Research Notes in Artificial Intelligence. Pitman, 1989.

[Cammarata *et al.*, 1983] S. Cammarata, D. McArthur, and R. Steeb. Strategies of cooperation in distributed problem solving. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 767–770, Karlsruhe, Federal Republic of Germany, August 1983.

[Conry *et al.*, 1992] S.E. Conry, K. Kuwabara, V.R. Lesser, and R.A. Meyer. Multistage negotiation in distributed constraint satisfaction. *IEEE Transactions on Systems, Man and Cybernetics—Special Issue on Distributed Artificial Intelligence*, January 1992.

[Davis and Smith, 1983] Randall Davis and Reid G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20:63–109, 1983.

[Durfee and Montgomery, 1990] Edmund H. Durfee and Thomas A. Montgomery. A hierarchical protocol for coordinating multiagent behaviors. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 86–93, Boston, Massachusetts, August 1990.

[Klein, 1991] Mark Klein. Supporting conflict resolution in cooperative design systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1379–1390, November/December 1991.

[Laasri *et al.*, 1992] B. Laasri, H. Laasri, S. Lander, and V. Lesser. Toward a general model of intelligent negotiating agents. *The International Journal on Intelligent Cooperative Information Systems*, 1992.

[Lander and Lesser, 1992a] Susan E. Lander and Victor R. Lesser. Customizing distributed search among agents with heterogeneous knowledge. In *Proceedings of the First International Conference on Information and Knowledge Management*, pages 335–344, Baltimore, Maryland, November 1992.

[Lander and Lesser, 1992b] Susan E. Lander and Victor R. Lesser. Negotiated search: Organizing cooperative search among heterogeneous expert agents. In *Proceedings of the Fifth International Symposium on Artificial Intelligence, Applications in Manufacturing and Robotics*, pages 351–358, Cancun, Mexico, December 1992.

[Lander, 1993] Susan E. Lander. *Distributed Search in Heterogeneous and Reusable Multi-Agent Systems.* PhD thesis, University of Massachusetts, Amherst, Massachusetts, 1993. In preparation.

[Lesser and Erman, 1980] Victor R. Lesser and Lee D. Erman. Distributed interpretation: A model and experiment. *IEEE Transactions on Computers*, C-29(12):1144–1163, December 1980.

[Neches *et al.*, 1991] Robert Neches, Richard Fikes, Tim Finin, Thomas Gruber, Ramesh Patil, Ted Senator, and William R. Swartout. Enabling technology for knowledge sharing. *AI Magazine*, 12(3):36–56, Fall 1991.

[Sathi and Fox, 1989] Arvind Sathi and Mark S. Fox. Constraint-directed negotiation of resource reallocations. In Les Gasser and Michael Huhns, editors, *Distributed Artificial Intelligence, Volume 2*, pages 163–193. Pitman, Morgan Kaufmann Publishers, 1989.

[Sycara *et al.*, 1991] K. Sycara, S. Roth, N. Sadeh, and M. Fox. Distributed constrained heuristic search. *IEEE Transactions on Systems, Man and Cybernetics*, Fall 1991.

[Sycara, 1985] Katia Sycara. Arguments of persuasion in labour mediation. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 294–296, Los Angeles, California, 1985.

[von Martial, 1992] Frank von Martial. *Coordinating Plans of Autonomous Agents.* Lecture Notes in Artificial Intelligence, Springer-Verlag, 1992.

[Werkman, 1992] Keith J. Werkman. Multiple agent cooperative design evaluation using negotiation. In *Proceedings of the Second International Conference on Artificial Intelligence in Design*, Pittsburgh, PA, June 1992.

[Yokoo *et al.*, 1992] Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara. Distributed constraint satisfaction for formalizing distributed problem solving. In *Proceedings of the Twelfth International Conference on Distributed Computing Systems*, Yokohama, Japan, June 1992.

[Zlotkin and Rosenschein, 1990] Gilad Zlotkin and Jeffrey S. Rosenschein. Negotiation and conflict resolution in non-cooperative domains. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, Massachusetts, July 1990.

APPENDIX C

Exploiting Meta-Level Information in a Distributed Scheduling System*

Daniel E. Neiman, David W. Hildum, Victor R. Lesser

Tuomas W. Sandholm
Computer Science Department
University of Massachusetts
Amherst, MA 01003
DANN@CS.UMASS.EDU

May 13, 1994

Abstract

In this paper, we study the problem of achieving efficient interaction in a distributed scheduling system whose scheduling agents may borrow resources from one another. Specifically, we expand on Sycara's use of resource texture measures in a distributed scheduling system with a central resource monitor for each resource type and apply it to the decentralized case. We show how analysis of the abstracted resource requirements of remote agents can guide an agent's choice of local scheduling activities not only in determining local constraint tightness, but also in identifying activities that reduce global uncertainty. We also exploit meta-level information to allow the scheduling agents to make reasoned decisions about when to attempt to solve impasses locally through backtracking and constraint relaxation and when to request resources from remote agents. Finally, we describe the current state of negotiation in our system and discuss plans for integrating a more sophisticated cost model into the negotiation protocol. This work is presented in the context of the Distributed Airport Resource Management System, a multi-agent system for solving airport ground service scheduling problems.

*This work was partly supported by DARPA contract N00014-92-J-1698 and NSF contracts CDA-8922572 and IRI-9208920. The content of this paper does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

1 Introduction

The problem of scheduling resources and activities is known to be extremely challenging [8, 7, 14, 11]. The complexity increases when the scheduling process becomes dependent upon the activities of other concurrent schedulers. Such interactions between scheduling agents arise when, for example, agents must borrow resources from other agents in order to resolve local impasses or improve the quality of a local solution. Distributed scheduling applications are not uncommon, for example, the classic meeting planning problem [13] can be considered as a distributed scheduling problem; the airport ground service scheduling (AGSS) problem we address in this paper is another; and similar problems may arise in factory floor manufacturing domains.

In distributed scheduling systems, problem-solving costs will likely increase because of the interaction among agents caused by the lending of resources. One method of increasing the quality of solutions developed by such multi-agent schedulers and minimizing the costs of backtracking is to allow agents to communicate abstracted versions of their resource requirements and capabilities to other agents. The use of this *meta-level* information allows the scheduling agents to develop models of potential interactions between their scheduling processes and those of other agents, where an interaction is defined as a time window in which the borrowing or lending of a resource might occur. We show how the identification of interactions affects the choice of scheduling heuristics, communication, and negotiation policies in a distributed scheduling system. We discuss our heuristics in the context of a specific testbed application, the Distributed Airport Resource Management System (DIS-ARM).

2 Related Work

The use of meta-level information to define the interactions between agents has been studied extensively by Durfee and Lesser via the use of partial global plans [5]. This work has been extended by Decker and Lesser [3, 4] to incorporate more sophisticated coordination relationships. According to this framework, we can view our detection of potential loan requests using texture measures to be an identification of *facilitating* relationships, and our modification of the scheduling algorithm as an attempt to exploit this perceived relationship. The formulation of distributed constraint satisfaction problems as distributed AI was described by Yakoo [16], however, this work concentrated more on the problem of distributed backtracking rather than on coordinating agents.

The problem of coordinating distributed schedulers has been studied extensively by Sycara and colleagues [15]. They describe a mechanism for transmitting abstractions of resource requirements (*textures*) between agents. Each agent uses these texture measures to form a model of the aggregate system demand for resources. This model is used to allocate resources using various heuristics. For example, a *least-constraining-value* heuristic is used to allocate resources based on the minimization of the probablity that the reservation would conflict with any other. For each type of resource, one agent is assigned the task of coordinating allocations and determining whether requests can be satisfied. All resources of a given type are considered interchangeable and the centralized resource monitor does not need to perform significant planning to choose the most suitable resource; instead, its role is simply to ensure that each resource is allocated to no more agents than can be served by that resource during any given time period.

We investigate a similar use of abstracted resource demands for a case in which centralized resource monitors are not possible since resources of the same type may possess unique characteristics, and agents possess proprietary information about local resources (such as current location and readiness). Agents may respond to a request for a resource either by immediately satisfying it with a reservation, denying it, or by performing local problem-solving actions to attempt to produce a suitable reservation.

In our domain, we have found that Sycara's texture measures alone do not convey sufficient information to allow satisfactory scheduling. Their texture measures consist of a demand profile for each resource which represents, for each time interval, the sum of probabilities that resource requests will overlap that interval. These probabilities are based on the assumption that reservations can occur at any time within the requested interval. Assignment of resources is then performed using these probabilities to implement a least-constraining-value heuristic.

These texture measures do not capture sufficient information regarding time-shift preferences of resource assignments within the specified interval. In our domain, resources may legally be assigned at any time within the interval between the earliest start time and the latest finish time, but for some activities, there exist strong preferences as to which end of the interval the assignment is biased. For example, when scheduling ground services for an airport, once a flight arrives, it is important to unload baggage as early as possible so that necessary transfers can be made to connecting flights. The shift preference can be determined by the assigning agent using domain knowledge, provided that it knows the nature of the task generating the request. Because this information is not captured in the texture measures, the heuristic described by Sycara, *et al.* is likely to lead to poor schedules within the airport ground service scheduling domain.

3 Overview: The Distributed Dynamic Scheduling System

In order to test our approach to solving distributed *resource-constrained scheduling problems* (RCSPs), we have designed a distributed version of a reactive, knowledge-based scheduling system called DSS (the Dynamic Scheduling System) [9]. DSS provides a foundation for representing a wide variety of real-world RCSPs. Its flexible scheduling approach is capable of reactively producing quality schedules within dynamic environments that exhibit unpredictable resource and order behavior. Additionally, DSS is equipped to manage the scheduling of shared tasks connecting otherwise separate orders, and handle RCSPs that involve mobile resources with significant travel requirements.

DSS is implemented as an agenda-based blackboard system [6, 1] using GBB (the Generic Blackboard System) [2]. It maintains a blackboard structure upon which a developing schedule is constructed, and where the sets of orders and resources for a particular RCSP are stored. A group of knowledge sources are provided for securing the necessary resource reservations. These knowledge sources are triggered as the result of developments on the blackboard, namely the creation and modification of the service goals attached to all resource-requiring tasks. Triggered knowledge sources are placed onto an agenda and executed in the order of their priority.

The Distributed Dynamic Scheduling System (DIS-DSS) maintains separate blackboard structures for each agent and provides communication utilities for transmitting requests and meta-level information between agents. Remote analogues of service goals, task structures, and other scheduling entities are created as needed to model the state of other agents. The information about other agents' schedules and commitments is incomplete and is limited to the content of goals, meta-level information, and those parts of the schedule to which the local agent itself has contributed.

The approach we have taken towards distributing DSS is to view each agent as representing an autonomous organization possessing its own resources. It is this autonomous nature of the organizations that is the rationale for distributing the resource allocation problem. Although a centralized architecture might produce more efficient solutions, real world considerations such as cost and ownership often lead to confederations in which information transfer regarding commitments and capabilities is limited. In this model, the primary relationship between agents is a commitment to exchange resources as needed and a willingness to negotiate with other agents to resolve impasses. This model of a decentralized group of agents performing independent tasks in a resource-constrained environment is similar to the architecture of Moehlman's Distributed Fireboss [10]. We distinguish our work from Moehlman's by our use of meta-level information to control the decision process by which agents choose to resolve impasses locally, through backtracking and constraint relaxation, or through requests to remote agents.

Because resources are owned by specific agents and possess unique characteristics regarding location and travel times that are known only to the owning agent, we can not define central resource monitors responsible for allocating each type of resource. This, again, distinguishes our approach from that of Sycara, et al. [15]. Agents requiring a resource must communicate directly with the agent owning a resource of that type and negotiate for its loan.

This architecture provides a rich domain for the study of agent coordination issues in a distributed environment; agents must be able to model the interactions of their tasks with those of neighboring agents closely enough to be able to determine which agents will be most likely to provide the desired resources at the lowest cost to both agents. This coordination requires local reasoning on the part of agents in order to determine how to cooperate efficiently with an acceptable level of communication and redundant computation.

3.0.1 Assumptions

In our work with DIS-DSS, we have made a number of assumptions about the nature of agents, schedules, and communication overheads.

- Agents are cooperative and will lend a resource if it is available.
- Agents will only request a resource from one agent at a time – this is to avoid the possibility of redundant computation and communication if multiple agents attempt to provide the resource cf. [12].
- Once agents have lent a resource to another agent, they will never renege on this agreement. This limits the ability of the system to perform global backtracking; we intend to eliminate this restriction in the next version of the system.
- Communication is asynchronous and can occur at any point during the construction of a local schedule; therefore requests may arrive before an agent has completely determined its own requirements for resources in the time window of interest.
- The cost of messages is largely in the processing and in the inherent delay caused by transmission – the amount of data within the message may be large, within limits.

3.0.2 Communication of Abstract Resource Profiles

Without information regarding other agents' abilities to supply missing resources, an agent may be unable to complete a solution, or may be forced to compromise the quality of its solution. To allow agents to construct a model of global system constraints and capabilities, we have developed a protocol for the exchange and updating of resource profiles: summarizations of the agent's committed resources, available resources, and estimated future demand.

Upon startup, each agent in DIS-DSS receives a set of orders to be processed. The agents examine these orders and generate an abstract description of their resource requirements for the scheduling period. This *bottleneck-status-list* consists of a list of intervals, with each interval annotated by a triple: resources in use, resources requested, and resources available. The request field of this triplet represents an abstraction of the agent's true resource requirements. Certain aspects of a reservation such as mobile resource travel times to the objects to be serviced, cannot be easily estimated in advance. The time intervals specified for each resource request are pessimistic, consisting of the earliest possible start time and latest possible finish times for the activity requesting that resource. The true duration of the task can be estimated by the scheduling agent using its domain knowledge regarding the typical time required to perform a task. We define the demand for a resource r performing task T in interval (t_j, t_k) to be:

$$\text{avg\_demand}(T, r, t_j, t_k) = \text{duration}(T, r)/(t_k - t_j)$$

Once resource abstractions have been developed for each resource type required (or possessed) by the agent, it transmits its abstractions to all other agents. Likewise, it receives abstractions from all agents. Once the agent has received communications from all other agents, it prepares a map of global resource requirements and uses it to generate a set of data structures called *lending possibilities.* Each lending possibility represents an interval in which some agent appears to have a shortfall in a resource. For each lending possiblity, the agent generates a list of possible lenders for that resource, based on the global resource map and its knowledge of its own resource requirements. These lending possibility structures are used to predict when remote agents may request resources and when the local agent may need to borrow resources. This information guides the agent's decision-making process in determining both when to process local goals and when and from whom to request resources.

3.1 The Distributed Airport Resource Management System

The Distributed Airport Research Management System testbed was constructed using DIS-DSS to study the roles of coordination and negotiation in a distributed problem-solver. DIS-ARM solves distributed AGSS problems where the function of each scheduling agent is to ensure that each flight for which it is responsible receives the ground servicing (gate assignment, baggage handling, catering, fuel, cleaning, etc.) that it requires in time to meet its arrival and departure deadlines. The supplying of a resource is usually a multi-step task consisting of setup, travel, and servicing actions. Each resource task is a subtask of the airplane servicing supertask. There is considerable parallelism in the task structure: many tasks can be done simultaneously. However, the choice of certain resource assignments can often constrain the start and end times of other tasks. For example, selection of a specific arrival gate for a plane may limit the choice of servicing vehicles due to transit time from their previous servicing locations and may limit refueling options due to the presence or lack of underground fuel tanks at that gate. For this reason, all resources of a specific type can not be considered interchangeable in the AGSS domain. Only the agent that owns the resource can identify all the current constraints on that resource and decide whether or not it can be allocated to meet a specific demand.

4 Exploiting Meta-level Information in DIS-DSS

In this section, we examine three areas in which meta-level abstractions of global resource requirements are exploited in DIS-DSS. We show how the goal rating scheme of an agent's blackboard-based scheduler is modified to satisfy the twin aims of scheduling based on global constraints and of planning activities in order to reduce uncertainty about agent interactions. We describe how communication of resource abstractions is based on models of agents' interests and the manner in which agents choose between local and remote methods of satisfying a request.

4.1 Scheduling using Texture Measures

Many scheduling systems divide processing into the categories of variable selection, the choice of the next activity to schedule, and value selection, the selection of a resource and time slot for that activity. In DIS-DSS, variable selection corresponds to the satisfaction of a particular resource request. Value selection is handled in DSS by a collection of opportunistic scheduling heuristics. We focus here on the problem of coordinating resource requests so that local variable-selection heuristics possess sufficient information to make informed decisions.

In many knowledge-based scheduling systems, the object of control is to arrange scheduling activities so that the most tightly constrained activities are scheduled first in order to reduce the need for backtracking. In a distributed system, we have an additional criterion: to schedule problem-solving activities in such a way that global uncertainty about certain tasks is reduced before decisions regarding those tasks are made. A scheduler may be uncertain of whether other agents will request a resource in a tightly constrained time period and whether other agents will be able to supply a needed resource. While the resource abstractions may indicate a loan request is likely, the duration of the loan and details of the resource's destination can only be determined once the request has been received. Likewise, details of the precise timing and duration of a loan can only be determined upon receipt of a remote reservation. We have added coordination heuristics to the agenda scheduler of DIS-DSS whose purpose is to promote problematic activities in each agent's scheduling queue so that their early execution will reduce uncertainty about global system requirements.

In the DIS-DSS blackboard-based architecture, tasks which require resources generate *service-goals*. Requests received from remote agents generate *remote-service-goals*. Each goal stimulates knowledge sources that act to secure an appropriate resource. The order of execution of knowledge sources depends on the rating of the stimulating goals. Goals are rated using a basic 'most-tightly-constrained-first' opportunistic heuristic. The goals are then stratified according to the following scheme, with the uppermost levels receiving the highest priority and contention within each level being resolved according to the basic rating heuristic.

1. Tightly constrained goals that may not be satisfiable locally or that can only be satisfied by a borrowing event *and* remote requests that do not overlap any local request.
2. Tightly constrained goals that can *only* be satisfied locally.
3. Goals representing requests from remote agents that overlap local goals.
4. Unconstrained or loosely constrained tasks.
5. Goals that *potentially* overlap with tasks of remote agents.

A goal g is considered to be tightly constrained in interval (t_j, t_k) if there exists a time within that interval such that for each resource type r that can satisfy g, the number of unreserved resources is less than the sum of the average demand for all outstanding goals.

$$\forall\ r\ s.t.\ \mathrm{Sat}(g, r)\ \exists\ t \in (t_j, t_k)\ s.t.$$

$$\mathrm{N}_{available}(r, t) < \sum_{g} \text{avg\_demand}(task(g), r, t_j, t_k)$$

A goal potentially overlaps with a task of a remote agent if there exists a lending-possibility data structure for that remote agent describing a potential shortfall within the time interval spanned by that goal for some resource type that could satisfy the goal.

The rationale for this goal ordering is as follows. Goals that can not be satisfied locally must be transmitted to remote agents. The transmission of a goal conveys considerably more information than is available in the resource texture profiles. The potential lending agent will therefore have more accurate information regarding the interval for which the resource is desired and the preferred shift preference for the reservation in that interval (early or late). Once it has received the goal, it will be able to make more informed decisions about the tightness of constraints for both the local and remote goals. If the agent is able to satisfy the remote goal, it will be able to update its resource demand curve and transmit it to other agents who may also have been potential lenders of that resource. For all these reasons, early transmittal and satisfaction of remote service goals is desirable.

Tightly constrained goals that potentially overlap remote requests are deferred until some overlapping goal arrives, or until a resource update arrives indicating that the remote agent no longer requires that resource, or until no other work is available for the agent to perform. By deferring goals until more information about interactions is available, the system can avoid making premature decisions while at the same time working on unrelated or less constrained tasks. Once a request arrives, conflicts for resources can be arbitrated according to which goal is most pressing and least conducive to backtracking and/or constraint relaxation.

There are a number of competing requirements for the rating and processing of remote service goals. One would like to process a remote service goal as soon as possible in order to return information to the requesting agent. At the same time, both local and remote service goals requesting the same type of resource should be rated according to the same constraint tightness heuristics. The goal rating function in DIS-DSS attempts to satisfy these requirements by prioritizing those remote service goals that do not overlap any local service goals and by mapping overlapping remote service goals onto the same priority level as those local goals that they overlap. Note that the "overlapping" relationship is transitive: if the priority of a goal is reduced while waiting for a remote request, any lower rated goal that overlaps that goal's time interval must also wait even though it may not directly overlap the interval of the potential remote request.

4.2 Guiding Communication using Texture Measures

Reducing communication costs is an important issue in distributed systems. For this reason, DIS-DSS agents use the lending possibility models of agent interactions to guide communication activities. When its resource requirements change, an agent transmits the information about the resource type only to those agents who, based on its local information, would be interested in receiving updates concerning that resource type. An agent with no surplus resources of a given type may not be interested if the local agent increases its need for a particular resource, likewise, an agent with a surplus of a particular resource may not need to be notified if an agent reduces its demand for that resource type. However, agents who possess shortfalls in a time interval for a particular type of resource will receive updates during processing whenever an agent increases the precision of its resource abstractions by securing or releasing a resource.

The use of local knowledge to guide communication episodes may lead to agents' knowledge of the global state of the system becoming increasingly out of date. The degree to which this should be allowed to happen is dependent upon the acceptable level of uncertainty in the system and the accuracy with which resource abstractions can be made.

4.3 Ordering Methods for Achieving Resource Assignments

In DSS, the process of securing a resource is achieved through a series of increasingly costly methods: assignment, preemption, and right shifting. These correspond roughly to request satisfaction, backtracking, and constraint relaxation. Preemption is a conservative form of backtracking in which existing reservations are preempted in favor of a more constrained task. Right shifting satisfies otherwise intractable requests by shifting the time interval of the reservation downstream (later) until a suitable resource becomes available. Because this method relaxes the latest finish time constraint, it has the potential to seriously decrease the quality of a solution. In the AGSS domain, for example, right shifting a reservation may result in late departures.

In DSS, methods are ordered according to increasing cost. In the distributed version of the system, the choice and ordering of methods is more complex. When an agent cannot immediately acquire a resource locally, it faces a decision: should it perform backtracking or constraint relaxation locally, communicating only when it has exhausted all local alternatives, or should it immediately attempt to borrow the resource from another agent? The decision-making process becomes even more difficult if we allow requests from remote agents to take precedence over local requirements such that agents may have to perform backtracking or constraint relaxation in order to satisfy a remote request. We consider this last decision process a form of *negotiation*, because it involves determining which of two agents should bear the cost of reduced solution quality and/or increased problem-solving effort.

In DIS-DSS, we use the lending possibility data structures to dynamically generate plans for achieving each resource assignment. When it appears that a remote agent will have surplus resources at the necessary time, then the agent will generate a request as soon as it becomes clear that the resource can not be secured locally. If, however, it appears that the resource is tightly constrained globally, the agent will choose to perform backtracking and/or constraint relaxation operations locally rather than engage in communication episodes that will probably prove futile.

One use of meta-information occurs during the planning for constraint relaxation. The scheduling agent attempts to minimize the magnitude of the right shift in order to reduce the effect of the constraint relaxation on the quality of the solution. To do this, the agent must determine whether the minimum right shift can be achieved locally or remotely. However, requiring agents to submit bids detailing their earliest reservations for a given resource would be a costly process. Instead, the agent uses the abstractions of remote resource availability to generate a threshold value for the right shift delay. If this value is less than the delay achieved through right shifting locally, the agent sequentially transmits the resource request to the appropriate remote agents. If a remote agent can provide a reservation with a delay of less than or equal to the threshold value, it immediately secures the resource. Otherwise, it returns the delay of the earliest possible reservation. If no reservation is found, the local agent sets the threshold to the earliest possible value returned by some remote agent. This new threshold is then compared to the current best local delay (which might have changed due to local scheduling while the remote requests were being processed). This process continues until a reservation is made or until the threshold becomes greater than the delay achievable by right shifting locally. Obviously, the better the initial estimate for the delay threshold, the less communication activities will be required.

The meta-information is also used to determine the order in which agents should be asked for resources, beginning with the agent(s) with the least tightly constrained resources.

5 Experimental Results

The performance of the mechanisms that we have developed for DIS-DSS were tested in a series of experiments using a single agent system as a basis for comparison. We used six scenarios designed to test the performance of the system in tightly constrained situations. The number of orders in each scenario ranged from 10 to 60 and a minimal set of resources was defined for each scenario. Each scenario was distributed for a three agent case. Orders were assigned to each agent on a round-robin basis such that each agent would perform approximately the same amount of work. Resources were distributed randomly so that in some cases each agent would possess all necessary resources while in other cases, borrowing from remote agents would be necessary.

We ran DIS-ARM on each scheduling scenario using the following configurations of the scheduler:

- The baseline case with a single agent.
- The 3 agent case with no use of meta-level information, and an opportunistic (most-tightly-constrained-variable-first) goal rating scheme
- The 3 agent case using the heuristic goal rating scheme incorporating meta-level information but requesting resources from remote agents only when all local methods have failed.
- The 3 agent case using heuristic goal rating, meta-level information, and dynamic reordering of resource acquisition methods to account for the probability of securing a goal either locally or remotely.

For each run, we recorded the average tardiness of the schedule, the number of failed goals (if any), the number of resource-securing methods tried, the number of requests, the number of satisfied remote service goals, and the number of communication episodes that occurred during problem solving. In each case, we assumed that communication costs were negligible in relation to problem-solving and that requests and resource constraint updates would be received on the simulation cycle immediately succeeding the one in which they were sent.

Because of the small number of test cases we have examined in our preliminary experiments, we present our results anecdotally. As expected, the distributed version of the scheduler always produces a schedule of somewhat lower quality than the centralized one. When the opportunistic scheduler of the centralized version is used for scheduling in a distributed environment, its lack of information about global constraints causes it to produce somewhat inferior results. The heuristic incorporating meta-level information consistently outperforms the opportunistic scheduler in terms of the number of tardy tasks. The opportunistic scheduler occasionally will produce a schedule with less total tardiness than the distributed algorithm. We interpret this as a trade-off between satisfying global requirements (by delaying certain goal satisfactions until remote information becomes available) and satisfying local requirements by producing needed results promptly. This is an interesting trade-off that we intend to study in depth. Attempting to always solve problems locally using preemption and constraint relaxation produced schedules with much greater delays than when agents dynamically determined when to request resources remotely based on the meta-level resource abstractions.

6 Conclusions and Future Work

The work we have performed with DIS-DSS is preliminary, but promising. Our results indicate that the idea of using meta-level information to schedule activities in order to reduce local uncertainty about global constraints results in better coordination between agents with a subsequent increase in goal satisfaction. We have also demonstrated that meta-level information can be successfully used to guide the choice between satisfying goals locally and remotely, and in optimizing the choice of agents from which to request resources.

Our experiments were performed with each agent's orders being defined statically before scheduling. This allowed the agents to develop a model of their predicted resource requirements before scheduling began. If we were to model a system in which orders changed dynamically, either due to equipment failures or timetable changes, we would expect the model of global resource requirements to become increasingly inaccurate. We would like to understand the implications of allowing jobs to arrive dynamically on the performance of a distributed system using meta-level information.

As well as continuing to explore the role of meta-level resource abstractions, we plan to use the DIS-DSS testbed to explore a number of important issues in distributed scheduling. One of our primary goals is to expand the idea of negotiation between agents that we have touched upon in this paper. Because the airport ground service scheduling domain represents a "real world" scenario, we are able to create a meaningful cost model involving not only the delay in each schedule, but the probable cost of that delay in terms of missed connections. By allowing agents to exchange this information when requesting resources, they will be able to more meaningfully weigh the importance of local tasks against the quality of the global solution.

References

[1] N. Carver and V. Lesser. The evolution of blackboard control architectures,. *Expert Systems with Applications- Special Issue on the Blackboard Paradigm and Its Applications*, 7(1):1–30, Jan–Mar 1994.

[2] D. D. Corkill, K. Q. Gallagher, and K. E. Murray. GBB: A generic blackboard development system. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 1008–1014, Philadelphia, PA., August 1986.

[3] Keith S. Decker and Victor R. Lesser. Generalizing the partial global planning algorithm. *International Journal of Intelligent and Cooperative Information Systems*, 1(2):319–346, June 1992.

[4] Keith S. Decker and Victor R. Lesser. Quantitative modeling of complex computational task environments. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 217–224, Washington, July 1993.

[5] E.H. Durfee and V.R. Lesser. Partial global planning: A coordination framework for distributed hypothesis formation. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(5):1167–1183, September 1991.

[6] Lee D. Erman, Frederick Hayes-Roth, Victor R. Lesser, and D. Raj Reddy. The hearsay-ii speech-understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys*, 12(2):213–253, June 1980.

[7] Mark S. Fox. *Constraint-Directed Search: A Case Study of Job-Shop Scheduling.* PhD thesis, Carnegie Mellon University, Pittsburgh PA, December 1983.

[8] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, New York, 1979.

[9] David W. Hildum. *Flexibility in a Knowledge-Based System for Solving Dynamic Resource-Constrained Scheduling Problems.* PhD thesis, Computer Science Dept., University of Massachusetts, Amherst, MA 01003, May 1994.

[10] Theresa A. Moehlman, Victor R. Lesser, and Brandon L. Buteau. Decentralized negotiation: An approach to the distributed planning problem. *Group Decision and Negotiation*, (2):161–191, 1992.

[11] Norman Sadeh. *Look-Ahead Techniques for Micro-Opportunistic Job Shop Scheduling.* PhD thesis, Carnegie Mellon University, Pittsburgh PA, March 1991.

[12] Tuomas Sandholm. An implementation of the contract net protocol based on marginal cost calculations. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 256–262, Washington, July 1993.

[13] Sandip Sen and Edmund Durfee. A formal analysis of communication and commitment in distributed meeting scheduling. In *Proceedings of the Twelfth Workshop on Distributed AI*, Hidden Valley, PA, May 1993.

[14] Stephen F. Smith, Mark S. Fox, and Peng Si Ow. Constructing and maintaining detailed production plans: Investigations into the development of knowledge-based factory scheduling systems. *AI Magazine*, 7(4):45–61, Fall 1986.

[15] K. Sycara, S. Roth, N. Sadeh, and M. Fox. Distributed constrained heuristic search. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1446–1461, November/December 1991.

[16] M. Yakoo, T. Ishida, and K. Kuwabara. Distributed constraint satisfaction for DAI problems. In *Proceedings of the 10th International Workshop on Distributed Artificial Intelligence*, October 1990.

APPENDIX D

Issues in Automated Negotiation and Electronic Commerce: Extending the Contract Net Framework

Tuomas Sandholm and Victor Lesser *
{sandholm, lesser}@cs.umass.edu
University of Massachusetts at Amherst
Computer Science Department
Amherst, MA 01003

Abstract

In this paper we discuss a number of previously unaddressed issues that arise in automated negotiation among self-interested agents whose rationality is bounded by computational complexity. These issues are presented in the context of iterative task allocation negotiations. First, the reasons why such agents need to be able to choose the stage and level of commitment dynamically are identified. A protocol that allows such choices through conditional commitment breaking penalties is presented. Next, the implications of bounded rationality are analyzed. Several tradeoffs between allocated computation and negotiation benefits and risk are enumerated, and the necessity of explicit local deliberation control is substantiated. Techniques for linking negotiation items and multiagent contracts are presented as methods for escaping local optima in the task allocation process. Implementing both methods among self-interested bounded rational agents is discussed. Finally, the problem of message congestion among self-interested agents is described, and alternative remedies are presented.

1 Introduction

The importance of automated negotiation systems is likely to increase [Office of Technology Assesment (OTA), 1994]. One reason is the growth of a fast and inexpensive standardized communication infrastructure (EDI, NII, KQML [Finin *et al.*, 1992], Telescript [General Magic, Inc., 1994] etc.), over which separately designed agents belonging to different organizations can interact in an open environment in real-time, and safely carry out transactions [Kristol *et al.*, 1994; Sandholm and Lesser, 1995d]. Secondly, there is an industrial trend towards *agile enterprises*: small, organizational overhead avoiding enterprises that form short term alliances to be able to respond to larger and more diverse orders than they individually could. Such ventures can take advantage of economies of scale when they are available, but do not suffer from diseconomies of scale. This concept paper explores the implications of performing such negotiations where agents are *self-interested* (SI) [1] and must make negotiation decisions in real-time with *bounded or costly computation resources.*

We cast such negotiations in the following domain independent framework. Each agent has a (possibly empty) set of tasks and a (possibly empty) set of resources it can use to handle tasks. These sets change due to domain events, e.g. new tasks arriving or resources breaking down. The agents can subcontract tasks to other agents by paying a compensation. This subcontracting process can involve breaking a task into a number of subtasks handled by different agents, or clustering a number of tasks into a supertask. A task transfer is profitable from the global perspective if the contractee can handle the task less expensively than the contractor, or if the contractor cannot handle it at all, but the contractee can. So, the problem has two levels: a *global task allocation problem*, and each agent's *local combinatorial optimization problem* defined by the agent's current tasks and resources. The goal of each agent is to maximize its *payoff* which is defined as its income minus its costs. Income is received for handling tasks, and costs are incurred by using resources to handle the tasks. We restrict ourselves to domains where the feasibility and cost of handling a task do not depend on what other agents do with their resources or how they divide tasks among themselves, but do depend on the other tasks that the agent has [2]. The global solution can be evaluated from a social welfare viewpoint according to the sum of the agents' payoffs.

Reaching good solutions for the global task allocation problem is difficult with SI agents, e.g. because they may not truthfully share all information. The problem is further complicated by the agents' bounded rationality: local decisions are suboptimal due to the inability

*This research was supported by ARPA contract N00014-92-J-1698. The content does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred. T. Sandholm also funded by a University of Massachusetts Graduate School Fellowship, Leo and Regina Wainstein Foundation, Heikki and Hilma Honkanen Foundation, and Ella and George Ehrnrooth Foundation.

[1] In domains where agents represent different real world organizations, each agent designer will want its agent to do as well as it can without concern for other agents. Conversely, some domains are inherently composed of benevolent agents. For example, in a single factory scheduling problem, each work cell can be represented by an agent. If the cells do not have private goals, the agents should act benevolently.

[2] Such domains are a superset of what [Rosenschein and Zlotkin, 1994] call Task Oriented Domains, and intersect their State Oriented and Worth Oriented Domains.

to precisely compute the value associated with accepting a task. This computation is especially hard if the feasibility and cost of handling a task depend on what other tasks an agent has. These problems are exacerbated by the uncertainty of an open environment in which new agents and new tasks arrive - thus previous decisions may be suboptimal in light of new information.

The original contract net protocol (CNP) [Smith, 1980] did not explicitly deal with these issues, which we think must be taken into account if agents are to operate effectively in a wide range of automated negotiation domains. A first step towards extending the CNP to deal with these issues was the work on TRACONET [Sandholm, 1993]. It provided a formal model for bounded rational (BR) self-interested agents to make announcing, bidding and awarding decisions. It used a simple static approximation scheme for *marginal cost*[3] calculation to make these decisions. The choice of a contractee is based solely on these marginal cost estimates. The monetary payment mechanism allows quantitative tradeoffs between alternatives in an agent's negotiation strategy. Within DAI, bounded rationality (approximate processing) has been studied with cooperative agents, but among SI agents, perfect rationality has been widely assumed, e.g. [Rosenschein and Zlotkin, 1994; Ephrati and Rosenschein, 1991; Kraus *et al.*, 1992]. We argue that in most real multiagent applications, resource-bounded computation will be an issue, and that bounded rationality has profound implications on both negotiation protocols and strategies.

Although the work on TRACONET was a first step towards this end, it is necessary—as discussed in the body of this paper—to extend in significant ways the CNP in order for bounded rational self-interested (BRSI) agents to deal intelligently with uncertainty present in the negotiation process. This new protocol represents a family of different protocols in which agents can choose different options depending on both the static and dynamic context of the negotiation. The first option we will discuss regards commitment. We present ways of varying the stage of commitment, and more importantly, how to implement *varying levels of commitment* that allow more flexible local deliberation and a wider variety of negotiation risk management techniques by allowing agents to back out of contracts. The second option concerns local deliberation. Tradeoffs are presented between negotiation risks and computation costs, and an approximation scheme for marginal cost calculation is suggested that dynamically adapts to an agent's negotiation state. The third set of options has to do with avoiding local optima in the task allocation space by *linking negotiation items* and by *contracts involving multiple agents.* The fourth set of options concerns *message congestion management.* We present these choices in terms of a new protocol for negotiation among BRSI agents, that, to our knowledge, subsumes the CNP and most—if not all—of its extensions.

[3]The *marginal cost* of adding a set of tasks to an agent's solution is the cost of the agent's solution with the new task set minus the cost of the agent's solution without it.

2 Commitment in negotiation protocols

2.1 Alternative commitment stages

In mutual negotiations, *commitment* means that one agent binds itself to a potential contract while waiting for the other agent to either accept or reject its offer. If the other party accepts, both parties are bound to the contract. When accepting, the second party is sure that the contract will be made, but the first party has to commit before it is sure. Commitment has to take place at some stage for contracts to take place, but the choice of this stage can be varied. TRACONET was designed so that commitment took place in the bidding phase as is usual in the real world: if a task is awarded to him, the bidder has to take care of it at the price mentioned in the bid. Shorter protocols (commitment at the announcement phase[4]) can be constructed as well as arbitrarily long ones (commitment at the awarding phase or some later stage).

The choice of commitment stage can be a static protocol design decision or the agents can decide on it dynamically. For example, the focused addressing scheme of the CNP was implemented so that in low utilization situations, contractors announced tasks, but in high utilization mode, potential contractees signaled availability—i.e. bid without receiving announcements first [Smith, 1980; Van Dyke Parunak, 1987]. So, the choice of a protocol was based on characteristics of the environment. Alternatively, the choice can be made for each negotiation separately before that negotiation begins. We advocate a more refined alternative, where agents dynamically choose the stage of commitment of a certain negotiation during that negotiation. This allows any of the above alternatives, but makes the stage of commitment a negotiation strategy decision, not a protocol design decision. The offered commitments are specified in *contractor messages* and *contractee messages*, Fig. 1.

2.2 Levels of commitment

In traditional multiagent negotiation protocols among SI agents, once a contract is made, it is binding, i.e. neither party can back out. In cooperative distributed problem solving (CDPS), commitments are often allowed to be broken unilaterally based on some local reasoning that attempts to incorporate the perspective of common good [Decker and Lesser, 1995]. A more general alternative is to use protocols with continuous levels of commitment based on a monetary penalty method, where commitments vary from unbreakable to breakable as a continuum by assigning a commitment breaking cost to each commitment separately. This cost can also increase with time, decrease as a function of acceptance time of the offer, or be conditioned on events in other negotiations or the environment. Using the suggested message types, the level of commitment can also be dynamically negotiated over on a per contract or per task set basis.

[4]With announcement phase commitment, a task set can be announced to only one potential bidder at a time, since the same task set cannot be exclusively awarded to many agents.

Among other things, the use of multiple levels of commitment allows:

- a low commitment search focus to be moved around in the global task allocation space (because decommitting is not unreasonably expensive), so that more of that space can be explored among SI agents which would otherwise avoid risky commitments[5],
- flexibility to the agent's local deliberation control, because marginal cost calculation of a contract can go on even after that contract has already been agreed upon,
- an agent to make the same low-commitment offer (or offers that overlap in task sets) to multiple agents. In case more than one accepts, the agent has to pay the penalty to all but one of them, but the speedup of being able to address multiple agents in committal mode may outweigh this risk,
- the agents with a lesser risk aversion to carry a greater portion of the risk. The more risk averse agent can trade off paying a higher price to its contractee (or get paid a lower price as a contractee) for being allowed to have a lower decommitting penalty, and
- contingency contracts by conditioning the payments and commitment functions on future negotiation events or domain events. These enlarge the set of mutually beneficial contracts, when agents have different expectations of future events or different risk attitudes [Raiffa, 1982].

The advantages of such a leveled commitment protocol are formally analyzed in [Sandholm and Lesser, 1995a], and are now reviewed. Because the decommitment penalties can be set arbitrarily high for both agents, the leveled commitment protocol can always emulate the full commitment protocol. Furthermore, there are cases where there is no full commitment contract among two agents that fulfills the participation constraints (agent prefers to agree to the contract as opposed to passing) for both agents, but where a leveled commitment contract does fulfill these constraints. This occurs even among risk neutral agents, for example when uncertainty prevails regarding both agents' future offers received, and both agents are assigned a (not too high or low, and not necessarily identical) decommitment penalty in the contract. Among risk neutral agents, this does not occur if only one of the agents is allowed the possibility to decommit (other agent's decommitment penalty is too high), or only one agent's future is uncertain. If the agents have biased information regarding the future, they may perceive that such a contract with a one-sided decommitment possibility is viable although a full commitment contract is not. In such cases, the agent whose information is biased is likely to take the associated loss while the agent with unbiased information is not.

Figure 1 describes the message formats of the new contracting protocol. A negotiation can start with either a

CONTRACTOR MESSAGE:
0. Negotiation identifier
1. Message identifier
2. In-response-to (message id)
3. Sender
4. Receiver
5. Terminate negotiation
6. Alternative 1
  6.1. Time valid through
  6.2. Bind after partner's decommit
  6.3. Offer submission fee
  6.4. Required response submission fee
  6.5. Task set 1
    (a) (Minimum) specification of tasks
    (b) Promised payment fn. to contractee
    (c) Contractor's promised commitment fn.
    (d) Contractee's required commitment fn.
  6.6. Task set 2
    ...
  6.i. Task set i-4
7. Alternative 2
  ...
j. Alternative j-5

CONTRACTEE MESSAGE:
0. Negotiation identifier
1. Message identifier
2. In-response-to (message id)
3. Sender
4. Receiver
5. Terminate negotiation
6. Alternative 1
  6.1. Time valid through
  6.2. Bind after partner's decommit
  6.3. Offer submission fee
  6.4. Required response submission fee
  6.5. Task set 1
    (a) (Maximum) specification of tasks
    (b) Required payment fn. to contractee
    (c) Contractor's required commitment fn.
    (d) Contractee's promised commitment fn.
  6.6. Task set 2
    ...
  6.m. Task set m-4
7. Alternative 2
  ...
n. Alternative n-5

PAYMENT/DECOMMIT MESSAGE:
0. Negotiation id
1. Message id
2. Accepted offer id
3. Acceptance message id
4. Sender
5. Receiver
6. Message type
   (payment/decommit)
7. Money transfer

Figure 1: *Contracting messages of a single negotiation.*

contractor or a contractee message, Fig. 2. A contractor message specifies exclusive alternative contracts that the contractor is willing to commit to. Within each alternative, the tasks can be split into disjoint task sets by the sender of the message in order for the fields (a) - (d) to be specific for each such task set - not necessarily the whole set of tasks. Each alternative has the following semantics. If the contractee agrees to handle all the task sets in a manner satisfying the minimum required task descriptions (a) (which specify the tasks and constraints on them, e.g. latest and earliest handling time or minimum handling quality), and the contractee agrees to commit to each task set with the level specified in field (d), then the contractor is automatically committed to paying[6] the amounts of fields (b), and can cancel the deal on a task set only by paying the contractee a penalty (c)[7]. Moreover, the contractor is decommitted

[5]For example, an agent can accept a task set and later try to contract the tasks in that set further separately. With full commitment, an agent needs to have standing offers from the agents it will contract the tasks to, or it has to be able to handle them itself. With the variable commitment protocol, the agent can accept the task set even if it is not sure about its chances of getting it handled, because in the worst case it can decommit.

[6]Secure money transfer can be implemented cryptographically e.g. by electronic credit cards or electronic cash [Kristol *et al.*, 1994].

[7]The "Bind after partner's decommit" (6.2) flag describes whether an offer on an alternative will stay valid according to its original deadline (field 6.1) even in the case where the contract was agreed to, but the partner decommitted by paying the decommitment penalty.

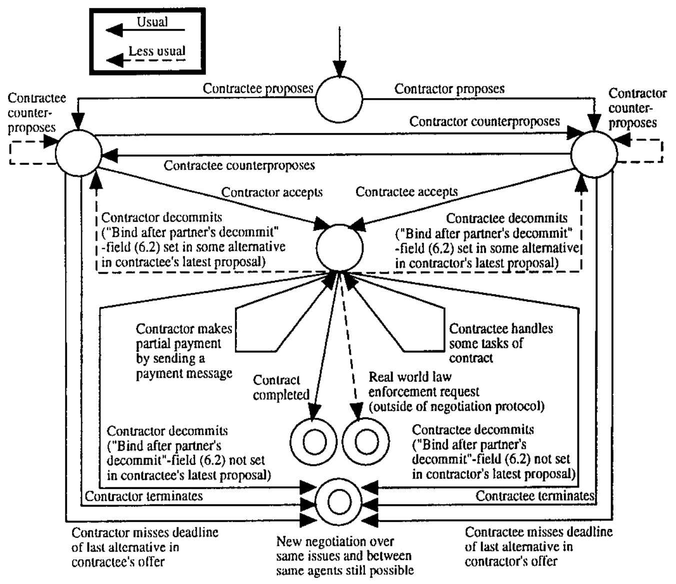
Usual
Less usual
Contractee proposes
Contractor proposes
Contractee counter-proposes
Contractor counter-proposes
Contractor counterproposes
Contractee counterproposes
Contractor accepts
Contractee accepts
Contractor decommits ("Bind after partner's decommit" -field (6.2) set in some alternative in contractee's latest proposal)
Contractee decommits ("Bind after partner's decommit" -field (6.2) set in some alternative in contractor's latest proposal)
Contractor makes partial payment by sending a payment message
Contractee handles some tasks of contract
Contract completed
Real world law enforcement request (outside of negotiation protocol)
Contractor decommits ("Bind after partner's decommit"-field (6.2) not set in contractee's latest proposal)
Contractee decommits ("Bind after partner's decommit"-field (6.2) not set in contractor's latest proposal)
Contractor terminates
Contractee terminates
Contractor misses deadline of last alternative in contractee's offer
New negotiation over same issues and between same agents still possible
Contractee misses deadline of last alternative in contractor's offer

Figure 2: *State transition diagram of a single negotiation.*

from all the other alternatives it suggested[8]. If the contractee does not accept any of the alternatives, the contractor is decommitted from all of them. Fields (b), (c) and (d) can be functions of time, of negotiation events, or of domain events, and these times/events have to be observable or verifiable by both the contractor and the contractee. A contractee can accept one of the alternatives of a contractor message by sending a contractee message that has task specifications that meet the minimal requirements (a), and payment functions that meet the required payment functions (b), and commitment functions (c) for the contractee that meet the required commitment functions, and commitment functions (d) for the contractor that do not exceed the contractor's promised commitment. A contractor message can accept one of the alternatives of a contractee message analogously. An agent can entirely terminate a negotiation by sending a message with that negotiation's identifier (field 0), and the terminate-flag (field 5) set.

Alternatively, the contractee can send a contractee message that neither accepts the contractor message (i.e. does not satisfy the requirements) nor terminates the negotiation. Such a message is a *counterproposal*, which the contractor then can accept, terminate the negotiation, or further counterpropose etc. *ad infinitum* [9]. The CNP did not allow counterproposing: an agent could bid to an announcement or decide not to bid. A contractor had the option to award or not to award the tasks according to the bids. Counterproposing among cooperative agents was studied in [Moehlman *et al.*, 1992; Sen, 1993]. Our counterproposing mechanism is one way of overcoming the problem of lacking truthful abstractions of the global search space (defined by the task sets and resource sets of all the agents) in negotiation systems consisting of SI agents.

There are no uncommittal messages such as announcements used to declare tasks: all messages have some commitment specification for the sender. In early messages in a negotiation, these commitment specifications can be too low for the partner to accept, and counterproposing occurs. Thus, the level and stage of commitment are dynamically negotiated along with the negotiation of taking care of tasks.

The presented negotiation protocol is a strict generalization of the CNP, and can thus always emulate it. Moreover, there are cases where this protocol is better than the CNP—due to reasons listed earlier. Yet, the development of appropriate negotiation *strategies* for this protocol is challenging—e.g. how should an agent choose commitment functions and payment functions?

2.3 Decommitting: replies vs. timeouts

The (6.1) field describes how long an offer on an alternative is valid. If the negotiation partner has not answered by that time, the sender of the message gets decommitted from that alternative. An alternative to these strict deadlines is to send messages that have the (b) field be a function of the time of response (similarly for (c) and (d) fields). This allows a contractor to describe a payment that decreases as the acceptance of the contractor message is postponed. Similarly, it allows a contractee to specify required payments that increase as the acceptance of the contractee message is postponed. This motivates the negotiation partner to respond quickly, but does not force a strict deadline, which can inefficiently constrain that agent's local deliberation scheduling. Both the strict deadline mechanism and this time-dependent payment scheme require that the sending or receival time of a message can be verified by both parties.

An alternative to automatic decommitment by the deadline is to have the negotiation partner send a negative reply (negotiation termination message) by the deadline. These forced response messages are not viable among SI agents, because an agent that has decided not to accept or counterpropose has no reason to send a reply. Sending reply messages also in negative cases allows the offering agent to decommit before the validity time of its offer ends. This frees that agent from considering the effects of the possible acceptance of that offer on the marginal costs of other task sets that the agent is negotiating over. This saved computation can be used to negotiate faster on other contracts. Thus, an agent considering sending a negative reply may want to send it in cases where the offering agent is mostly negotiating with that agent, but not in cases, where the offering agent is that agent's competing offerer in most other negotiations.

[8]Another protocol would have offers stay valid according to their original specification (deadline) no matter whether the partner accepts, rejects, counterproposes, or does none of these. We do not use such protocols due to the harmfully (Sec. 3) growing number of pending commitments.

[9]An agent that has just (counter)proposed can counterpropose again (dotted lines in Fig. 2). This allows it to add new offers (that share the "In-response-to"-field with the pending ones), but does not allow retraction of old offers. Retraction is problematic in a distributed system, because the negotiation partner's acceptance message may be on the way while the agent sends the retraction.

3 Implications of bounded rationality

Interactions of SI agents have been widely studied in microeconomics [Kreps, 1990; Varian, 1992; Raiffa, 1982] and DAI [Rosenschein and Zlotkin, 1994; Ephrati and Rosenschein, 1991; Kraus *et al.*, 1992; Durfee *et al.*, 1993], but perfect rationality of the agents has usually been assumed: flawless deduction, optimal reasoning about future contingencies and recursive modeling of other agents. Perfect rationality implies that agents can compute their marginal costs for tasks exactly and immediately, which is untrue in most practical situations. An agent is bounded rational, because its computation resources are costly, or they are bounded and the environment keeps changing—e.g. new tasks arrive and there is a bounded amount of time before each part of the solution is used [Garvey and Lesser, 1994; Sandholm and Lesser, 1994; Zilberstein, 1993; Simon, 1982; Good, 1971]. Contracting agents have the following additional real-time pressures:

- A counteroffer or an acceptance message has to be sent by a deadline (field 6.1) - otherwise the negotiation terminates, Fig. 2. If the negotiation terminates, the agent can begin a new negotiation on the same issues, but it will not have the other agent's commitment at first.
- Sending an outgoing offer too late may cause the receiving agent to make a contract on some of the same tasks with some other agent who negotiated earlier—thus disabling this contract even if the offer makes the deadline. In case this deadline abiding offer is an acceptance message—as opposed to a counteroffer—the partner has to pay the decommitment penalty that it had declared.
- The (b)-(d) fields can be functions of response time, Fig. 1. An agent may get paid less for handling tasks (or pay more for having tasks handled) or be required to commit more strongly or receive a weaker commitment from the negotiation partner if its response is postponed.
- The agent's cost of breaking commitments (after a contract is made) may increase with time.

This problem setup leads to a host of local deliberation scheduling issues. An agent has to decide *how much computation* it should allocate to refine its marginal cost estimate of a certain task set. With a bounded CPU, if too much time is allocated, another agent may win the contract before the reply is sent, or not enough time remains for refining marginal costs of other task sets. If too little time is allocated, the agent may make an unbeneficial contract concerning that task set. If multiple negotiations are allowed simultaneously, the agent has to decide on *which sets of tasks* (offered to it or potentially offered by it) its bounded computation should be focused—and *in what order*. It may want to ignore some of its contracting possibilities in order to focus more deliberation time to compute marginal costs for task sets of some selected potential contracts. So, there is a tradeoff of getting more exact marginal cost estimates and being able to engage in a larger number of negotiations.

The CNP did not consider an agent's risk attitude toward being committed to activities it may not be able to honor, or the honoring of which may turn out unbeneficial. In our protocol, an agent can take a risk by making offers while the acceptance of earlier offers is pending. Contracting during pending commitments speeds up the negotiations because an agent does not have to wait for results on earlier commitments before carrying on with other negotiations. The work on TRACONET formalized the questions of risk attitude in a 3-stage (announce-bid-award) full-commitment protocol, and chose a risk taking strategy where each agent ignored the chances of pending commitments being accepted in order to avoid computations regarding these alternative future worlds. This choice was static, but more advanced agents should use a risk taking strategy where negotiation risk is explicitly traded off against added computation regarding the marginal cost of the task set in the alternative worlds, where different combinations of sent pending offers are accepted.

There is a tradeoff between accepting or (counter) proposing early on and waiting:

- A better offer may be received later.
- Waiting for more simultaneously valid offers enables an agent to identify and accept synergic ones: having more options available at the decision point enables an agent to make more informed decisions.
- Accepting early on simplifies costly marginal cost computations, because there are fewer options to consider. An option corresponds to an item in the power set of offers that an agent can accept or make.
- By waiting an agent may miss opportunities due to others making related contracts first.

An agent should anticipate future negotiation and domain events in its strategy [Sandholm and Lesser, 1995b].[10] It suffices to take these events into account in marginal cost estimation: this will cause the agent to anticipate with its domain solution. The real marginal cost of a task set is the difference in the *streams of payments and domain costs* when an agent has the task set and when the agent does not have it. This marginal cost does not necessarily equal the cost that is acquired statically at contract time (before the realization of unknown future negotiation events and domain events) by taking the difference of the cost of the agent's optimal solution with the task set and the optimal solution without it. Furthermore, for BR agents, the marginal cost may change as more computation is allocated to the solution including the task set or the solution without it. In general, the marginal cost of a task set depends on which other tasks the agent has. Therefore, theoretically, the marginal cost of a task set has to be computed in all of the alternative future worlds, where different combinations of pending,

[10]The agent can believe that domain events occur to the agent society according to some distribution and that in steady state these events will affect (directly or by negotiation) the agent according to some distribution. E.g. the agent assumes that future tasks end up in its task set according to a distribution. On another level, an agent can try to outguess the other agents' solutions so that it can use the others marginal costs as a basis for its own marginal cost calculation. On a third level, the agent can model what another agent is guessing about yet another agent, and so on *ad infinitum*. There is a tradeoff between allocating costly computation resources to such recursive modeling and gaining domain advantage by enhanced anticipation.

to-be-sent, and to-be-received offers have been accepted, different combinations of old and to-occur contracts have been broken by decommitting (by the agent or its partners), and different combinations of domain events have occurred. Managing such contingencies formally using probability theory is intractable: costs of such computations should be explicitly traded off against the domain advantage they provide. An agent can safely ignore the chances of other agents decommitting only if the decommitment penalties are high enough to surely compensate for the agent's potential loss. Similarly, an agent has to ignore its decommitting possibilities if its penalties are too high. The exponential number of alternative worlds induced by decommitting options sometimes increases computational complexity more than the benefit from the gradual commitment scheme warrants. Moreover, the decommitting events are not independent: chains of decommitting complicate the management of decommitment probabilities. Thus, decommitment penalty functions that increase rapidly in time may often be appropriate for BR agents.

Because new events are constantly occurring, the deliberation control problem is stochastic. An agent should take the likelihood of these events into account in its deliberation scheduling. The performance profile of the local problem solving algorithm should be conditioned on features of the problem instance [Sandholm and Lesser, 1994], on performance on that instance so far [Sandholm and Lesser, 1994; Zilberstein, 1993], and on performance profiles of closely related optimizations (related calculations of marginal costs). These aspects make exact decision theoretic deliberation control infeasible: approximations are required. The need for this type of deliberation control has not, to our knowledge, been well understood, and analytically developing a domain independent control strategy that is instantiated separately (using statistical methods) for each domain would allow faster development of more efficient automated negotiators across multiple domains.

4 Linking negotiation items

In early CNP implementations, tasks were negotiated one at a time. This is insufficient, if the cost or feasibility of carrying out a task depend on the carrying out of other tasks: there may be local optima, where no transfer of a single task between agents enhances the global solution, but transferring a larger set of tasks simultaneously does. The need for larger transfers is well known in centralized iterative refinement optimization [Lin and Kernighan, 1971; Waters, 1987], but has been generally ignored in automated negotiation. TRACONET extended the CNP to handle task interactions by having the announcer *cluster tasks into sets to be negotiated atomically*. Alternatively, the bidder could have done the clustering by counterproposing. Our protocol generalizes this by allowing either party to do the clustering, Fig. 1, at any stage of the protocol.

The equivalent of large transfers can be accomplished by smaller ones if the agents are willing to take risks. Even if no small contract is individually beneficial, the agents can sequentially make all the small contracts that sum up to a large beneficial one. Early in this sequence, the global solution degrades until the later contracts enhance it. When making the early commitments, at least one of the two agents has to risk taking a permanent loss in case the partner does not agree to the later contracts. Our protocol decreases such risks as much as preferred by allowing breaking commitments by paying a penalty. The penalty function may be explicitly conditioned on the acceptance of the future contracts, or it may specify low commitment for a short time during which the agent expects to make the remaining contracts of the sequence.

Sometimes there is no task set size such that transferring such a set from one agent to another enhances the global solution. Yet, there may be a beneficial *swap* of tasks, where the first agent subcontracts some tasks to the second and the second subcontracts some to the first. Swaps can be explicitly implemented in a negotiation protocol by allowing some task sets in an alternative (Fig. 1) to specify tasks to contract in and some to specify tasks to contract out. In the task sets added to implement swaps, "Minimum" in field (a) should be changed to "Maximum" and vice versa. In field (b), "Promised payment fn. to contractee" should be changed to "Required payment fn. from contractee" and "Required payment fn. to contractee" should be changed to "Promised payment fn. from contractee". Alternatively, in protocols that do not explicitly incorporate swaps, they can be made by agents taking risks and constructing the swap as a sequence of one way task transfer contracts. Here too, the decommitment penalty functions can be conditioned on later contracts in the sequence or on time to reduce (or remove) risk.

5 Mutual vs. multiagent contracts

Negotiations may have reached a local optimum with respect to each agent's local search operators and mutual contract operators (transfers and swaps of any size), but solution enhancements would be possible if tasks were transferred among more than two agents, e.g. agent A subcontracts a task to C and B subcontracts a task to C. There are two main ways to implement such deals[11]:

1. Explicit multiagent contracts. These contract operators can be viewed as atomic operators in the global task allocation space. First, one agent (with an incomplete view of the other agents' tasks and resources) has to identify the beneficiality of a potential multiagent contract. Alternatively, the identification phase can be implemented in a distributed manner. Second, the protocol has to allow a multiagent contract. This can be done e.g. by circulating the contract message among the parties and agreeing that the contract becomes valid only if every agent signs.

2. Multiagent contracts through mutual contracts. A multiagent contract is equivalent to a sequence of mutual contracts. In cases where a local optimum with respect to mutual contracts has been reached,

[11]Sathi et al. [Sathi and Fox, 1989] did this by having a centralized mediator cluster several announcements and bids from multiple agents into atomic contracts. That is unreasonable if decentralization is desired.

the first mutual contracts in the sequence will incur losses. Thus, one or more agents have to incur risk in initially taking unbeneficial contracts in unsure anticipation of more than compensatory future contracts. Our protocol provides mechanisms for decreasing this risk, either by conditioning the decommitment penalty functions on whether the contracts with other agents take place, or by choosing the penalties to be low early on and increase with time. In the limit, the penalty is zero (theoretically possibly even negative) for all contracts in the sequence if some contract in it is not accepted. The problem with contingency contracts is just the monitoring of the events that the contract (penalty) is contingent on: how can the contractee monitor the contractor's events and vice versa?

Sometimes an agent can commit to an unprofitable early contract in the sequence without risk even with constant high decommitting penalties. E.g. if an agent has received committal offers on two contracts, it can accept both without risk—assuming that decommitment penalties for the two senders are so high that they will not decommit. Even though the agent may have some offers committed simultaneously, the likelihood of having all the necessary offers committed simultaneously decreases as the number of mutual contracts required in the multiagent contract increases. Sometimes there is a loop of agents in the sequence of mutual contracts, e.g. say that the only profitable operator is the following: agent A gives task 1 to agent B, agent B gives task 2 to agent C, and agent C gives task 3 to agent A. In such cases it is impossible to handle the multiagent contract as separate mutual contracts without risk (without tailoring the decommitment penalty functions). A negotiating agent should take the possibilities of such loops into account when estimating the probabilities of receiving certain tasks, because the very offering or accepting of a certain task may directly affect the likelihood of getting offers or acceptances for other tasks.

6 Message congestion: Tragedy of the commons

Most distributed implementations of automated contracting have run into message congestion problems [Smith, 1980; Van Dyke Parunak, 1987; Sandholm, 1993]. While an agent takes a long time to process a large number of received messages, even more messages have time to arrive, and there is a high risk that the agent will finally be saturated. Attempts to solve these problems include focused addressing [Smith, 1980], audience restrictions [Van Dyke Parunak, 1987; Sandholm, 1993] and ignoring incoming messages that are sufficiently outdated [Sandholm, 1993]. Focused addressing means that in highly constrained situations, agents with free resources announce availability, while in less constrained situations, agents with tasks announce tasks. This avoids announcing too many tasks in highly constrained situations, where these announcements would seldom lead to results. In less constrained environments, resources are plentiful compared to tasks, so announcing tasks focuses negotiations with fewer messages. Audience restrictions mean that an agent can only announce to a subset of agents which are supposedly most potential.

Focused addressing and audience restrictions are imposed on an agent by a central designer of the agent society. Neither is viable in open systems with SI agents. An agent will send a message whenever it is beneficial to itself even though this might saturate other agents. With flat rate media such as the Internet, an agent prefers sending to almost everyone who has non-zero probability of accepting/counterproposing. The society of agents would be better off by less congested communication links by restricted sending, but each agent sends as long as the expected utility from that message exceeds the decrease in utility to that agent caused by the congesting effect of that message in the media. This defines a *tragedy of the commons* [Turner, 1992; Hardin, 1968] (n-player prisoners' dilemma). The tragedy occurs only for low commitment messages (usually early in a negotiation): having multiple high commitment offers out simultaneously increases an agent's negotiation risk (Sec. 2.2) and computation costs (Sec. 3).

The obvious way to resolve the tragedy is a use-based communication charge. Another is mutual monitoring: an agent can monitor how often a certain other agent sends low commitment messages to it, and over-eager senders can be punished. By mutual monitoring, audience restrictions can also be implemented: if an agent receives an announcement although it is not in the appropriate audience, it can directly identify the sender as a violator. Our protocol allows an agent to determine in its offer (field 6.4) a processing fee that an accepting or counterproposing agent has to submit in its response (field 6.3) for the response to be processed. This implements a self-selecting dynamic audience restriction that is viable among SI agents.

7 Conclusions

We introduced a collection of issues that arise in automated negotiation systems consisting of BRSI agents. Reasons for dynamically chosen commitment stage and level were given and a protocol that enables this was presented. The need for explicit local deliberation scheduling was shown by tradeoffs between computation costs and negotiation benefits and risk. Linking negotiation items and multiagent contracts were presented as methods to avoid local optima in the global task allocation space, and their implementation among BRSI agents was discussed. Finally, message congestion mechanisms for SI agents were presented.

Negotiations among BRSI agents also involve other issues (detailed in [Sandholm and Lesser, 1995b] due to limited space here) such as: insufficiency of the Vickrey auction to promote truth-telling and stop counterspeculation, usefulness of long term strategic contracts, tradeoffs between enforced and unenforced contracts [Sandholm and Lesser, 1995d], and knowing when to terminate the negotiations when an optimum with respect to the current tasks and resources has been reached or when further negotiation overhead outweighs the associated benefits. Coalition formation among BRSI agents has been studied in [Sandholm and Lesser, 1995c].

References

[Decker and Lesser, 1995] Keith Decker and Victor R Lesser. Designing a family of coordination algorithms. In *1st International Conference on Multiagent Systems*, San Fransisco, CA, June 1995.

[Durfee *et al.*, 1993] Edmund H Durfee, J Lee, and Piotr J Gmytrasiewicz. Overeager reciprocal rationality and mixed strategy equilibria. In *AAAI-93*, pages 225–230, Washington DC, July 1993.

[Ephrati and Rosenschein, 1991] Eithan Ephrati and Jeffrey S Rosenschein. The clarke tax as a consensus mechanism among automated agents. In *AAAI*, pages 173–178, Anaheim, CA, 1991.

[Finin *et al.*, 1992] Tim Finin, Rich Fritzson, and Don McKay. A language and protocol to support intelligent agent interoperability. In *Proc. of the CE & CALS Washington '92 Conference*, June 1992.

[Garvey and Lesser, 1994] A Garvey and V Lesser. A survey of research in deliberative real-time artificial intelligence. *Real-Time Systems*, 6:317–347, 1994.

[General Magic, Inc., 1994] General Magic, Inc. Telescript technology: The foundation for the electronic marketplace, 1994. White paper.

[Good, 1971] Irving Good. Twenty-seven principles of rationality. In V Godambe and D Sprott, editors, *Foundations of Statistical Inference*. Toronto: Holt, Rinehart, Winston, 1971.

[Hardin, 1968] G Hardin. The tragedy of the commons. *Science*, 162:1243–1248, 1968.

[Kraus *et al.*, 1992] Sarit Kraus, Jonathan Wilkenfeld, and Gilad Zlotkin. Multiagent negotiation under time constraints. Univ. of Maryland, College Park, Computer Science TR-2975, 1992.

[Kreps, 1990] David M Kreps. *A course in microeconomic theory*. Princeton University Press, 1990.

[Kristol *et al.*, 1994] David M Kristol, Steven H Low, and Nicholas F Maxemchuk. Anonymous internet mercantile protocol. 1994. Submitted.

[Lin and Kernighan, 1971] S Lin and B W Kernighan. An effective heuristic procedure for the traveling salesman problem. *Operations Research*, 21:498–516, 1971.

[Moehlman *et al.*, 1992] T Moehlman, V Lesser, and B Buteau. Decentralized negotiation: An approach to the distributed planning problem. *Group Decision and Negotiation*, 2:161–191, 1992.

[Office of Technology Assesment (OTA), 1994] Office of Technology Assesment (OTA). Electronic enterprises: Looking to the future, 1994.

[Raiffa, 1982] H. Raiffa. *The Art and Science of Negotiation*. Harvard Univ. Press, Cambridge, Mass., 1982.

[Rosenschein and Zlotkin, 1994] Jeffrey S Rosenschein and G Zlotkin. *Rules of Encounter*. MIT Press, 1994.

[Sandholm and Lesser, 1994] Tuomas W Sandholm and Victor R Lesser. Utility-based termination of anytime algorithms. In *ECAI Workshop on Decision Theory for DAI Applications*, pages 88–99, Amsterdam, The Netherlands, 1994. Extended version: Univ. of Mass. at Amherst, Comp. Sci. Tech. Report 94-54.

[Sandholm and Lesser, 1995a] Tuomas W Sandholm and Victor R Lesser. Advantages of a leveled commitment contracting protocol. Univ. of Mass. at Amherst, Comp. Sci. Tech. Report, 1995. In preparation.

[Sandholm and Lesser, 1995b] Tuomas W Sandholm and Victor R Lesser. Automated contracting among self-interested bounded rational agents. Technical report, University of Massachusetts at Amherst Computer Science Department, 1995. In preparation.

[Sandholm and Lesser, 1995c] Tuomas W Sandholm and Victor R Lesser. Coalition formation among bounded rational agents. In *Proc. 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, 1995.

[Sandholm and Lesser, 1995d] Tuomas W Sandholm and Victor R Lesser. Equilibrium analysis of the possibilities of unenforced exchange in multiagent systems. In *Proc. 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, Montreal, 1995.

[Sandholm, 1993] Tuomas W Sandholm. An implementation of the contract net protocol based on marginal cost calculations. In *Proc. 11th National Conference on Artificial Intelligence (AAAI-93)*, July 1993.

[Sathi and Fox, 1989] A Sathi and M Fox. Constraint-directed negotiation of resource reallocations. In Michael N. Huhns and Les Gasser, eds., *Distributed Artificial Intelligence*, vol. 2 of *Research Notes in Artificial Intelligence*, ch. 8, pages 163–193. Pitman, 1989.

[Sen, 1993] Sandip Sen. *Tradeoffs in Contract-Based Distributed Scheduling*. PhD thesis, Univ. of Michigan, 1993.

[Simon, 1982] Herbert A Simon. *Models of bounded rationality*, volume 2. MIT Press, 1982.

[Smith, 1980] Reid G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, December 1980.

[Turner, 1992] Roy M Turner. The tragedy of the commons and distributed ai systems. In *Proceedings of the 12th International Workshop on Distributed Artificial Intelligence*, pages 379–390, May 1992.

[Van Dyke Parunak, 1987] H Van Dyke Parunak. Manufacturing experience with the contract net. In Michael N. Huhns, editor, *Distributed Artificial Intelligence*, Research Notes in Artificial Intelligence, chapter 10, pages 285–310. Pitman, 1987.

[Varian, 1992] Hal R Varian. *Microeconomic analysis*. New York: W. W. Norton, 1992.

[Waters, 1987] C D Waters. A solution procedure for the vehicle-scheduling problem based on iterative route improvement. *Journal of the Operational Research Society*, 38(9):833–839, 1987.

[Zilberstein, 1993] Shlomo Zilberstein. *Operational rationality through compilation of anytime algorithms*. PhD thesis, University of California, Berkeley, 1993.